MASTER IN COMPUTER SCIENCE

Computer sketching: how software tools can support creative practices?

Miche, Frédéric

Award date:
1999

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année académique 1998-1999

Computer Sketching : How Software Tools Can Support Creative Practices ?

Frédéric Miche

Mémoire présenté en vue de l'obtention du grade de 'Maître en Informatique'

Abstract

Creative thought is something occurring with little if any conscious awareness. But this does not mean that we cannot figure out how creativity works. We believe that to show creativity is to explore and transform the conceptual spaces of the relevant domains. Artists raised needs and reproaches about software technology we listed in a typology. Generally speaking, we can say that current software applications do not meet their demands – which reinforce creators' negative attitude towards technology. However, some of them have understood the potential for producing arresting and highly creative graphic products through computer-based tools. We see in this document the importance of the stage at which the designers are admitted in the process, the importance to listen to the practitioners of a domain and effectively take into consideration the context of action in which the software application will take place. We review traditional software engineering techniques and propose a better way of developing applications – named by some computer and designer professionals 'software design'. It should lead to effectively used pieces of software, contributing to enhance the user's experience and to reduce the hindrances. The Visual Assistant is a computer-based sketching tool using software design principles. It is aimed at supporting the creative practices of theatre professionals, educators and students. We believe in its adequacy to their practices and describe which of the above needs and claims expressed by creative practitioners are addressed by the Visual Assistant and which are not.

Résumé

La créativité est une chose se produisant presqu'inconsciemment. Ceci n'empêchant pas de comprendre comment elle fonctionne. Nous pensons que faire preuve de créativité consiste à explorer et transformer les espaces conceptuels des domaines concernés. Des artistes ont exprimé leurs besoins et des reproches concernant les logiciels – ces premiers sont repris dans une typologie. De manière générale, nous pouvons dire que les logiciels actuels ne rencontrent pas leurs demandes, et partant, renforcent leur attitude négative vis-à-vis de la technologie. Cependant, certains ont bien compris le potentiel des ordinateurs à produire des créations graphiques étonnantes et hautement créatives. Nous examinons dans ce mémoire l'importance du stade à partir duquel les designers sont admis dans le processus, l'importance d'écouter les praticiens d'un domaine particulier et d'effectivement prendre en considération le contexte d'action dans lequel le logiciel prendra place. Nous critiquons les techniques traditionnelles dites de software engineering et proposons une meilleure approche du développement d'applications informatiques – approche nommée 'software design' par certains professionnels de l'informatique et du design. Cette approche devrait conduire à des logiciels effectivement utilisés, contribuant à améliorer l'expérience vécue par l'utilisateur et à réduire les entraves. Le Visual Assistant est un outil informatique particulier permettant d'esquisser des idées ; cet outil a été développé en respectant les principes de software design. Il est destiné à venir en soutien des pratiques créatives des professionnels du théâtre, aux enseignants et étudiants. Nous sommes persuadés de son adéquation avec leurs pratiques et décrivons lesquels des besoins et reproches, exprimés ci-avant par les personnes utilisant la créativité dans leurs activités, sont abordés par le Visual Assistant et lesquels ne le sont pas.

Keywords

Computer sketching, context, creative practices, design, designer, domain of action, human experience, human-centred design, prototyping, sketching, software design, software engineering, stage, theatre, user experience, Visual Assistant, world of action.

Foreword

In the first place, I would like to thank Jacques Berleur and Colin Beardon for having proposed such an interesting subject of my training period and the master's degree dissertation. I would also like to thank Nic Earl and Ben Salem, members of the NVRCAD (the Networked Virtual Reality Centres for Art & Design) at the ESAD (Exeter School of Art & Design, University of Plymouth), for their everyday help and support as well as Dean, staff member of the ESAD and 'poor' housemate of Ben.

My thanks also to Sylvia Maddock (my landlady), David, 'Poppy' and my housemates for welcoming me so warmly in their house. My stay in Exeter (England) will always remain the best memory of my first twenty-four years thanks to the students (and friends) of the MATTA department, their tutor Jeremy Diggle, his assistant Mike Lawson-Smith and the rest of the staff at the ESAD, my friends at the ESAD and at the University of Exeter, the people I met during the French lessons at the famous 'The imperial' pub.

Back to Belgium, Colin Beardon and Jacques Berleur kept giving me their advises and helped me to write the present master's degree dissertation. Thanks again to them and to the people who proof read this master's degree dissertation (Jacques Berleur and my Australian cousin Karen Davey). Thanks to my family and my sister in particular for the lending of her laptop. I would also like to thank my fellow students and my hall of residence mates (all my friends) for their moral support and for the many times we played sports, went to the cinema or ate together. This really contributed to keep each of us going on with the writing of our master's degree dissertations. At last, my thanks go to my girlfriend Mariam who helped me to finish this work when it started to become 'lengthy'. She motivated me in the last moments of the writing.

MICHE Frédéric
Master's Degree of Computer Science,
5th Year

Master's degree dissertation

Introduction

Main advisor : Jacques BERLEUR, s.j. (FUNDP) Training advisor : Colin BEARDON (ESAD)

Introduction

In the domain of theatre, there is a long creative process of putting on a play even before the performance in public. This performance is the visible part of the iceberg, as some might say. This process starts with the writing of the play and the design of the visual aspects, that is the design of the settings.

Computer-based tools are increasingly used in numerous domains of our everyday modern. However, their utilisation in the creative domains is not so common use. They are sometimes used in fashion design, industrial product design, image or animation production, 3D modelling for artistic purposes, and a few others.

What is the use of computers in the domain of theatre practices ? The situation is quite similar to the one prevailing in the domain of creative practices. Computer consoles help in controlling the stage lighting system and other technical effects (sounds, for instance). People also work on computers to write the plays through the use of word processors. We think that is about all.

We believe that the use of computer-based tools can help to produce arresting and highly creative works of art, even in the domain of theatre in which the plays are not immediate outputs from a computer – unlike some advertising images or multimedia CD-ROMs. The first stage of any creative process makes absolutely necessary to work with ill-defined ideas and does not require the details right from the start. As a consequence, we believe in the necessity to develop sketching computer-based tools to support the work of theatre producers rather than in the use of complex 3D software packages that can lead to pernicious practices and hinder creativity.

In response to that, the Visual Assistant has been developed to help the theatre practitioners during the concept formation phase. It is a computer-based sketching tool that allows rapid prototyping of stage sets, offering users to feel free to explore new ideas, new designs in a short time. Unlike 3D software packages, it renders images with a rough quality rather than with full details.

The degree training served to develop a PC version of the Visual Assistant package currently running on the Macintosh platform. The need of a PC version is resulting from

the prevalence, from the supremacy of this platform nowadays, from the ever increasing use of PCs in any situation of work and leisure.

We do not merely translated the code of the Macintosh version into a programming language running on PC platforms. We used the Macintosh version as a specification document, as an application with functionalities, with particular behaviours to reproduce on PC. Besides, the idea was that I should come to understand the objectives of the Visual Assistant in terms of support to the creative practitioners.

One of the objectives of this master's degree dissertation is to see how we could develop effectively 'usable' software packages for creative practitioners, including people working in the domain of theatre. For this purpose, we first tried to clarify the notion of creativity as far as possible. We then looked at the relationship between artists, designers and computer technology, and we determined their needs and reproaches in terms of software technology. As part of that objective, we consider as relevant to measure the ability for tools to draw near the theatre practitioners and will therefore question software engineering methods with regard to software design principles. Software design is a discipline offering human-centred and action-centred distinctive features that could reveal themselves as useful for developing applications aimed at supporting design practices for theatre. At last, through this master's degree dissertation, we would like to show the importance of computer-based sketching tools and to bring the Visual Assistant face to face with the demands and reproaches of the creative practitioners and see if the Visual Assistant has attained its objectives.

MICHE Frédéric
Master's Degree of Computer Science,
5th Year

Master's degree dissertation

Chapter One

What is creativity ?

Main advisor : Jacques BERLEUR, s.j. (FUNDP) Training advisor : Colin BEARDON (ESAD)

Chapter One

What is creativity ?

Introduction

Before addressing the way software tools can help in supporting the creative practices – especially in the field of theatre – we found relevant to first try to define what creativity is – or better understand what it is about. Our search to understand how it could work – if it is ever possible to do so – will mostly be based on the work of Margaret A. Boden[1].

As a cognitive scientist, she presents an aim common to many researchers in the field of Artificial Intelligence (AI for short). She mainly focuses on trying to reproduce human-like creative practices inside computer machinery. For this purpose, she investigated the mind to figure out how it works in terms of creative practices and gave a first definition, an explanation of what creativity is. Her vision of creativity is one theory among several others : in no way it is 'the' theory about the mechanisms of creativity – it is just the insight into creativity's maze we needed.

Margaret A. Boden's account of creativity

An unconscious process

What is the so-called creativity ? We could ask it of some creative practitioners, inventors or scientists – creativity is at the heart of their works. Or to some psychologists whose concerns is to unveil the mystery wrapping the mechanisms of creativity. But they could not tell us much about creativity. All we could learn about it is that for some of them, it is a mystery that no scientific theory will ever explain. When others mention intuition, they mean that the intuition process is responsible for choosing and combining the right ideas which results in one that can be called 'creative'. To say it differently,

creativity is something very unpredictable, happening unexpectedly making it very difficult to give an explanation of the way it works. We have no conscious awareness of how it arises. But as [Boden, 1994, p.75] reminds us, creativity is not the only human unconscious process : "*vision, language, common-sense reasoning*" are also concerned. So why do the mechanisms of creativity seem more difficult, more 'taboo', to study than our vision for instance ? Maybe we should have a look in the dictionary to help us understand the phenomenon...

> **Creativity** : "*the power of creation, invention*" [Grand Robert, 1993].
>
> **Creation** : "*the act of bringing into being or forming out of nothing*" [Grand Robert, 1993; Boden, 1992, p.1].

According to this definition, "*creativity seems not only beyond any scientific understanding* – as said above – *but even impossible*" [Boden, 1994, p.75]. 'Out of nothing' – it is as if God was behind every 'creation' made by a human being. If we take a look at the definition of the adjective 'creative', we can find more helpful clues.

> **Creative** : "*producing or using new and effective ideas, results, etc.*" or "*someone who is creative is very imaginative and good at making things, painting, etc.*" [Longman, 1995].

The second meaning is more pragmatic but does not help a lot in understanding how creativity works. We learn from it that artists, inventors, etc. are commonly showing, making use of creativity. More interesting is the first meaning of the adjective. Creativity is, according to it, the generation of (new) ideas or the "*novel combination of old ideas*" [Boden, 1994, p.75]. And the more improbable the combination of ideas, the more surprising it is. This combination of ideas, as [Muller, 1993] points out, is also the result of a number of influences, either our personal past experience, our conversations with other people, our observations of events ("*e.g., on TV*"), our personal background, context ("*e.g., in relation to other problems*"), and culture.

[1] [Boden, 1994]

Senses of creativity

As part of the process of calling an idea 'creative', this idea is given some value according to its degree of 'interest'[2]. Among these valuable ideas, [Boden, 1992, p.32] distinguish between two "*senses of creativity : P-creativity and H-creativity*" – 'P' stands for 'psychological' and 'H' for 'historical'. She says that "*a valuable idea is P-creative if the person in whose mind it arises could not have had it before; it does not matter how many times other people have already had the same idea. By contrast, a valuable idea is H-creative if it is P-creative and no one else, in all human history, has ever had it before*".

The act of attributing first-time novelties to individuals is the concern of art and science historians. And this attribution is definitely not a particularly precise, or rigorous process since there are many unrelated factors involved in the fact that these evidences were brought to the historians' attention.

Creativity is a matter of novel ideas. [Boden, 1994, p.76] says that some ideas are creative but that others can be "*surprising in a deeper way* [if] *they concern novel ideas that not only 'did not' happen before but 'could not' have happened before*". Making this distinction should help us to distinguish between "*mere first-time newness and radical novelties*". The former can easily be explained based on what Chomsky said about linguistics : we use a finite set of rules – called 'grammar' – to generate sentences endlessly. These ones can have the following feature : they can be generated for the first time. However, the generative system (the grammar) is the same for all the sentences (within the same language); thus we can say that they 'could' have happened before even tough they 'did not' – someone could have potentially generated such a sentence before. Producing them is not to do something P-creative.

So, what is a radically novel, creative idea ? Consider a merely novel idea. You can produce it using a specific generative system and recognise/describe it with a specific system. Both systems are based on the same set of rules that are used to generate/describe other, familiar ideas. "*A radically creative idea cannot*" use such a familiar, pre-existing set of rules [Boden, 1994, p.78]. The reason is that one cannot link up any familiar idea with the generative/descriptive system since it is radically new – no one could have used that system to produce a novel idea.

[2] See below Can we measure creativity ? for an explanation of this scoring process.

Conceptual spaces

What is a 'conceptual space' ? And what do we need that concept for ? [Boden, 1994, p.79] uses the notion of conceptual spaces to refine her insight into creativity. She defines a conceptual space as "*the generative system underlying a domain of thinking and defining a certain range of possibilities*", that is, the things, concepts we can produce using that generative system. It is "*the organising principles that unify and give structure to a given domain of thinking*". The latter can be an artistic genre and to depict it, an artist needs an internal model of it. This internal model is the conceptual space of that artistic genre.

According to her, we build "*mental representations*" of these spaces with respect to the domain of thinking concerned (poetry, sculpture, architecture, chemistry, etc.), as we map geographical territories. And we use these maps to explore and possibly change those conceptual spaces. Such conceptual explorations often lead to novelties and she perceives it as the way creativity works – although you have to bear in mind that some novelties can be worthless as well. However, changing those conceptual spaces appear to her as a deeper form of creativity.

She distinguishes between two sorts of conceptual explorations and two sorts of changes. Some explorations lead us to discover parts of the relevant conceptual space that we had not noticed before – that is the first kind of exploration that can be done – while other explorations, "*by contrast*", take us to the limits of the conceptual space. Changes of the conceptual spaces could occur while exploring those limits. When we say 'could occur', it is actually us who change the contours of those spaces.

Before getting back to the kind of changes that can be made, let us consider an example of each type of exploration possible. In her book, [Boden, 1992, p.49] used an example taken from a classic of the English literature. "*When Dickens described Scrooge as 'a squeezing, wrenching, grasping, scraping, clutching, covetous old sinner', he was exploring the space of English grammar. He was reminding the reader (and himself) that the rules of grammar allow us to use any number of adjectives before a noun. Usually, we use only two or three; but we may, if we wish, use seven (or more). That possibility already existed, although its existence may not have been realised by the reader*". That example merely shows us something unpreviously noticed about a specific conceptual space.

When one manages to identify a specific limitation of a conceptual space, the path to wonder how this limitation could be overcome is not long. And "*to overcome a limitation in a conceptual space, one must change it in some way*" [Boden, 1994, p.80]. There are two kinds of changes possible. If you are not exploring the limits of a conceptual space (if you are in a superficial dimension of that space), any change you make is a relatively small change – which is called a 'tweak'. A tweak is like "*opening a door to an unvisited room in an existing house*" [Boden, 1994, p.80]. The first-time novelty resulting from that tweak can either be a worthless idea or a valuable one, that is to say, a creative idea.

On the other hand, you have larger changes that Margaret A. Boden named 'transformations' as they happen in more fundamental dimensions of our conceptual spaces. Such a change deeply modifies the structure of the generative/descriptive system so that no one can link up any familiar idea with it : this resulting system is the one mentioned above which leads to the production of a radically creative idea. A large change (transformation) is "*more like the instantaneous construction of a new house, of a kind fundamentally different from (albeit related to) the first*" [Boden, 1994, p.80].

The moment seems appropriate to introduce another description of creativity as parallels can be drawn between this definition and the account of Margaret A. Boden. This definition, from F.D. Peat, is given in [Muller, 1993]. "*Human creativity involves a free-flowing play of the mind in which new ideas constantly surface and interact with each other. Ideas are like patterns in a kaleidoscope which move and transform until some new pattern swings into perception*". The point of interest lies in the use of the same concepts as Margaret A. Boden : when she says 'the generation of ideas', he says 'ideas surfacing'; when she says 'combination of ideas', he says 'ideas interact with each other'. The ideas which move and transform in a kaleidoscope until something new swings into perception, this reminds us of Margaret A. Boden's exploration and transformation of the conceptual spaces, which can lead to novelties.

Can we measure creativity ?

No, not by applying a numeral scale. Creativity is not something you can measure the way we do for evaluating distances or chemicals concentration, for instance. However, you cannot escape the scoring phenomenon when it comes to estimating the value of an idea and to say whether it is creative or worthless.

Like we said above : as part of the process of calling an idea 'creative', this idea is given some value according to its degree of 'interest'. What we did not say is that this scoring process is different according to the culture : in different places and / or at different times, people will not value ideas in the same ways as others. What was creative for our ancestors is not creative anymore for us as the recognition/descriptive/generative systems have been used to produce many ideas which are now familiar to us. Using those systems does not lead to produce interesting, valuable novelties now.

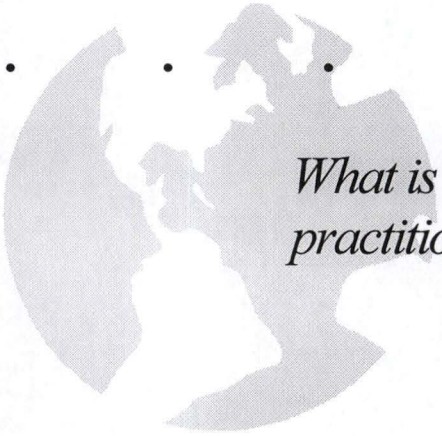
Conclusion

We endeavoured to explain what creativity is not to dispel the paradox, mystery and sense of wonder surrounding creative practices but to better appreciate the richness of creative thought thanks to a scientific approach. Creative thought is something occurring unexpectedly, with little if any conscious awareness. But this does not mean that we cannot figure out how creativity works. This knowledge of the mechanisms of creativity does not necessarily help us to control and orientate our creativity. More important than distinguishing between P- and H-creativity, we know that to show creativity is to explore and transform the spaces of expressive skills, the conceptual spaces of the relevant domains. Such enlightenment should help us to determine the effectiveness of the Visual Assistant as a computer-based tool to support creative practices.

MICHE Frédéric
Master's Degree of Computer Science,
5th Year

Master's degree dissertation

Chapter Two



What is the relationship between creative practitioners and computers ?

Main advisor : Jacques BERLEUR, s.j. (FUNDP) Training advisor : Colin BEARDON (ESAD)

Chapter Two

What is the relationship between artists and computers ?

Introduction

The subject of this master's degree dissertation is, in a few words, 'computer sketching' and 'creative practices'. In order to study how some software design methods should be devised or modified to produce software tools aimed at supporting the creative practices of artists and designers[1], we need to investigate the relationship between artists/designers and computer systems. This should help us to identify the needs and working practices of artists and designers and determine if the current software applications meet their demands. To say it another way, purpose-built new software applications designed for specific users and areas of use should reflect a critical analysis of the relationship of digital technologies to creativity and the design of contemporary software interfaces. In Chapter Four, we shall point out the needs – determined hereafter – to which the Visual Assistant answers.

The relationship of 'artists'[2] to computer systems

An initial investigation was conducted to better understand how creative practitioners such as artists or designers use computer systems. The results of the interviews carried out as a preparation for this study as well as the preliminary analysis can be found in [Beardon, 1997] – which has influenced the writing of this section. These first conclusions yielded an interesting insight into their relationship with computer systems.

Autonomy and institutions

One of the significant contexts that influence the work of artists and designers is the extent to which they are tied to an institutional framework. A work of an autonomous

[1] Also referred to as 'creative practitioners'.
[2] We should say 'creative practitioners' as this notion include designers as well.

nature features the control over the choice of the resources and the tools and put the artist or the designer in relation with various institutions – which gives them a feeling of greater independence. Working with a single institution is usually perceived as being more coercive since they are in closer relation with market orientations – though not in the majority of cases. For some artists and designers, working at an educational institution is seen "*as providing* [them] *with a level of resourcing that permits a degree of autonomy in* [their] *professional work*" [Beardon, 1997].

Very few studies which explore the links between institutions and individual artists/designers and the way their work is influenced by those institutions have been conducted so far. As a consequence, we could hardly "*come closer to understanding the institutional contexts within which* [the group of workers interviewed] *operates*" [Beardon, 1997]. So what follows is more focused on the links between creative practitioners and computer systems.

Concept formation and sketching

Several interviewees have stressed the fact that the computer still has no good software tools to start using it during the early stage of design, that is in the 'concept formation', or the 'initial idea generation'. One can argue that one of the reasons could lie in the very few suitable input devices, but we would rather like to put forward an issue that seems deeper to us. As an architect says, "*the order is inverted : you no longer follow the age-old method which used to consist of working up a rough idea, going on to the sketches – introducing precision step by step – and arriving at the details at the end of the process. The detail is required right from the start, during the preliminary phase : the 'natural' order is inverted*" [Beardon, 1997]. Because the 'initial idea generation' takes place out of the computer, a new stage in the process of creation appeared : that is the act of transferring the early work onto a computer system. Then, the artist or designer can continue the work on that digital medium, able to cope with the details once the first stage is over.

So far, there have been no tools which could relegate the need of details to the end of the creation/design process and this was "*a major obstacle to good design*" [Beardon, 1997]. Such 'sketching' tools would permit to quickly develop several ideas in parallel before selecting one – or a few – of them, all this without having to bear heavy costs. And

this is a feature ("*the ability to explore ideas*", "*new forms of exploration*" [Beardon, 1997]) of computer systems that creative practitioners would really desire. Theatre producers / writers would use such sketching software to make quick designs of each stage of their plays. Thus enabling them to better view how the play fits together.

The new forms of exploration mentioned above relate to the exploration of the conceptual spaces, as we saw in Chapter One. The adjective 'new' refers to the attempts to discover parts we had not noticed before as well as to reach and explore the limits of these spaces of creative practices and, subsequently, to overcome the same limits by changing the conceptual spaces themselves. Sketching tools represented as software seem to support the exploration but also the tweaks and transformations (the two sorts of changes distinguished by Margaret A. Boden and described in Chapter One) that can lead to creative or radically creative novelties.

Overcoming space and time limits

An artist interviewed mentioned that she had a sort of 'ease-of-contact' with computers. She stated that it could originate from her systematic way of working that she has always had – even before coming to use a computer for doing fine art print making. In her words, "she *found the transition to working with a computer 'quite natural'* " [Beardon, 1997]. We could draw from this that using a computer requires that you develop – unless it was already embodied – a sense of 'rigour' to help you to cope with the one of computer machineries. As set out in Chapter One, a conceptual space defines the range of possibilities in a given domain of thinking. Developing such a sense of rigour is actually reorganising the internal model that an artist/designer has of a certain artistic genre to include the use of computers in the production of works of art. Many artists find computer systems frustrating since they cannot feel the same sense of freedom, the same lack of coercion as in the traditional working practices, i.e. when they work without a computer.

New and other skills should be developed if one wants to take advantage of using a computer in order to take control of physical space. Before mentioning those skills, let us explain what 'the control of physical space' is. Many artists or designers face problems of space with a studio which is too small especially when they have to store – and retrieve – numerous and/or large documents of all kinds : painting canvas, blueprints, sculptures, notes, drawings, models, etc. Thus the use of a computer system is seen as a possibility to

store intelligently all those documents : instead of classifying according to physical features such the size, it is now possible to make that classification according to the content.

Such a way of proceeding is of a higher benefit for the creative practitioner : it helps saving physical space and it allows to store documents in specific contexts that would not have been possible without the use of a computer file system. In other words, using adequately the computer filing system should allow to create contextual meanings, thus making the design concept an organisational reality : all the varied documents stored in their specific context contribute to allow discerning the design in progress as a whole. Thanks to the context within which documents are stored, connections are suggested between several of them and it allows to rapidly gather material in order to publish a catalogue for an exhibition, among other examples.

"*The ability to reduce all working objects to a small rectangular window through which, with good organisation, everything is accessible is seen as a particular benefit*" [Beardon, 1997]. The point is that you should proceed '*with good organisation*', sorting out files and folders. Another skill is the ability to use finite, small windows on all kinds of large documents without being hindered while working. This is one of the reasons why many architects or designers refuse to handle blueprints and models on a computer screen.

However, using a computer is not of a benefit to every artist or designer. As one of them stated, some creative practitioners can become progressively less sensitive to space and light because of little to no mobility. By staring at a computer screen, just moving the fingers to type on the keyboard or move the mouse, they become 'indifferent to place and light'. According to her, this reflects in the work as the use of light become almost neutral. It is as if some parts of the concerned conceptual spaces were not explored anymore.

Some artists and designers see the Internet as a medium to "*overcome the limits of space and time*" [Beardon, 1997]. They do not need to be in physical contact anymore, to be at the same place, at the same time to exchange ideas and views, to collaborate on works of art or mount (actual or virtual) exhibitions. In the same way "*the Japanese tradition of 'Renga' makes poems pass round to enlarge them*", some artists "*make images pass round on the net, this being a sign of the advent of a televirtual art*" [Quéau, 1993, pp.114-115].

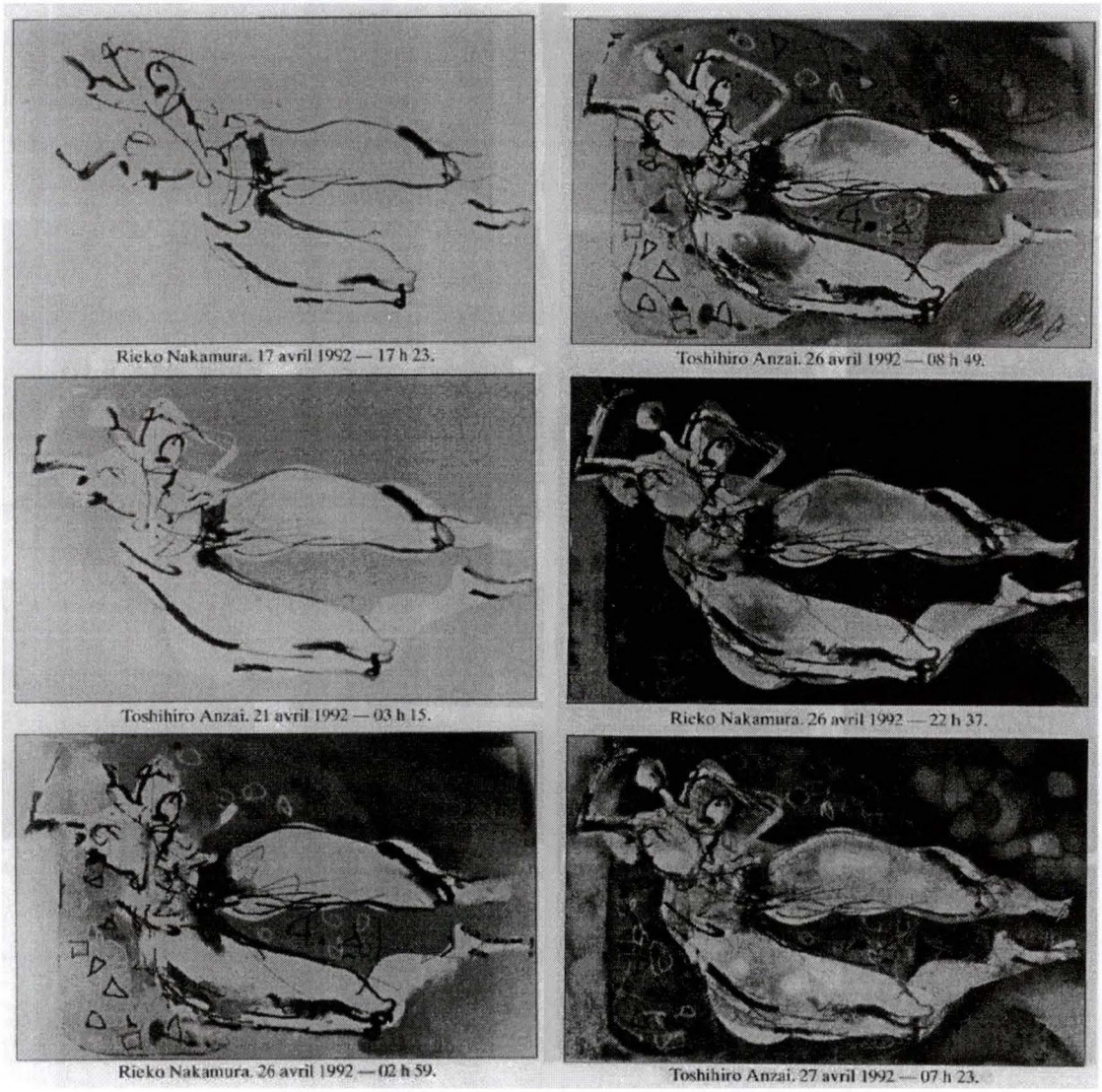


Fig. 2.1 : televirtual art [Quéau, 1993, pp.114-115]

They like the 'UNDO' command since it effectively offers users a means to reverse the effect of time by "*turning it back one unit or more*", thus "*returning the designed object to its previous state*" [Beardon, 1997]. This has the enormous advantage of allowing artists and designers to try several ideas, actions and later come back if they prove unsatisfactory. We would like to add that the use of sketching tools come and strengthen the tendency to explore several new ideas without adding regrets when the best thing to do is to discard some sketches, as we shall see in Chapter Four about Visual Assistant.

Metaphor : an obstacle to observe the marginal behaviour

In the previous section, we saw that computers can help to overcome some limits of the traditional creative practices. However, computers can sometimes be more of a hindrance than a help. Metaphors are one of those obstacles.

What is a metaphor ? It is the attempt to emulate historical expertise and/or existing professional skills in interfaces. As a matter of fact, such attempt hinders the creative practitioner from discovering, exploring the (unexpected) potential of a package. The reason, in the terms of Chapter One, is that the metaphor links any idea – developed by the artist/designer with the piece of software – to a familiar generative system, thus hindering the generation of a novel idea that cannot be linked to any familiar set of rules. Trying to design an interface reflecting a present – maybe even past and outdated – practice may be counter-productive as it could lead to trap artist/designer users whose creative practices are attempts to go beyond current understanding, to search for the unexpected, to unveil the potentialities.

To illustrate the above-mentioned point, let us quote a furniture designer that uses 3-D packages : "*having formed a conceptual model of the main functionality of the package, one then begins to observe software behaviour that is unexpected*" [Beardon, 1997]. Traditional software design methods do not take such practices into consideration, leading some users (the creative practitioners) not to use those traditionally designed applications.

What happens when the metaphor "*embodied in the package becomes an obstacle to understanding this marginal behaviour*" ? According to that same furniture designer, "*the response is to set limits on one's use of the package and to begin to look for a new product, which starts the process over again* (see the previous paragraph for that process)" [Beardon, 1997].

A bunch of complementary techniques

"*The computer is one of several techniques for experiencing and there is a developed sense* [among the interviewees at least] *of when it is appropriate to use it and when not to*" [Beardon, 1997]. What matters most is to stay as close as possible to the actual experience of real techniques (i.e. collage, model making, drawing and sketching, experimenting with colours, exploring spaces, etc.). Therefore, there are limited chances that these will be

replaced by only one tool – the computer, as it is. However, the computer can serve as a complement – and not as an exact replica – to those traditional artistic techniques.

They appreciate...

During the interviews carried out, these artists and designers mentioned many other interesting facts, most of them being the reasons for which they like using computer systems.

More than once, individuals have stated that computers could offer them the accuracy they needed in their work, the ability to edit and print multiple versions of their works. They also appreciate the fact that computers allow hybrid ways of working : with or without the computer according to the kind of experience, the results they are looking for. As a furniture designer said it, "*he does not consider the computer as an alternative for the whole of design work, but rather suggests that designers should be able to use the most appropriate process for the task at hand*" [Beardon, 1997].

Simplicity. An important feature in respect to the time it takes to get to know a package well. This learning phase can last up to "*2 years*" for some applications – the blame probably resides in the reversed order within which details are required from the very beginning of the design process. Hence their preference for "*packages that present themselves initially as quite simple, but reveal depths of complexity as you get to used them*" [Beardon, 1997]; interfaces that are not "*cluttered*", with no floweries.

They find the power of interactivity very attractive. Such feature of the computer "*can be used to change the locus of meaning away from the artist and towards the viewer. With interactivity, the viewer becomes indispensable*" (she or he takes part in the act of giving a meaning to the work of art) "*and the art work is the performance/experience*" [Beardon, 1997] and not the product itself anymore.

Typology

In the following typology, we have listed concerns taken from the results of the interviews exposed above. In Chapter Four, we shall see which of the concerns the Visual Assistant answers.

In the relation between artists/designers and computers, there should be…

- A feeling of greater independence in the choice of tools;
- Suitable software tools to use during the concept formation. These tools should offer the possibility to sketch, and consequently to explore, ideas rather then requiring details right from the start. A computer-based sketching tool should allow to assess ideas and, therefore, enhance the chances of success of a creation;
- Simplicity is often lacking in many software applications. There is a preference for "*packages that present themselves initially as quite simple, but reveal depths of complexity as you get to used them*";

To use computers, artists and designers need…

- A sense of good organisation and rigour;
- The ability to use finite windows on all sorts of large documents;
- To consider computers as a complement to – and not to exactly reproduce or substitute for – traditional artistic techniques offering hybrid ways of working;

Drawbacks…

- A loss of sensitivity to space and light;
- (repeated) A sense of good organisation and rigour;
- (repeated) The ability to use finite windows on all sorts of large documents;
- Metaphors can be perceived as obstacles to the observation of the software applications' marginal behaviour.

Advantages…

- Use of Internet to overcome space and time limits and collaborate on works of art or exhibitions;

- Great use of the UNDO command as it offers a means to reverse the effect of time;
- Computers offer artists and designers possibilities to edit and print multiple versions of their works;
- Interactivity is a very attractive feature of computers;
- Use of the computer to reduce storing space and intelligently store documents in a digital form;

Conclusion

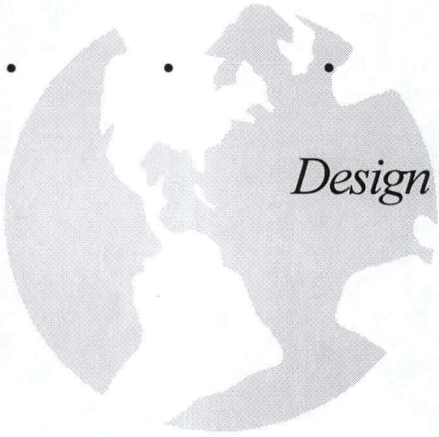
In this chapter, we analysed the needs and reproaches of artists and designers and drawn up a typology of them to later see which are addressed by the Visual Assistant. Generally speaking, we can say that current software applications do not meet their demands. Hence the funding for developing the Visual Assistant in order to fill the gap in terms of (sketching) tools available for use during the concept formation[3]. The lacks of features supporting creativity in many software applications, the qualities required for artists to use computers and the drawbacks of such use contribute to the reinforcement of artists' negative attitude towards the technology. However, some have understood the potential for producing arresting and highly creative graphic products through computer-based tools.

[3] See Chapter Four.

8MICHE Frédéric
Master's Degree of Computer Science,
5th Year

Master's degree dissertation

Chapter Three



Design of software for creative practices

Main advisor : Jacques BERLEUR, s.j. (FUNDP) Training advisor : Colin BEARDON (ESAD)

Chapter Three

Design of software for creative practices

Introduction

"*Our aim as not to make the computer the subject of the creative act*[1]*, but to develop computer tools that support the human exercise of design skills. Such tools could have a number of benefits : allowing ideas to be expressed in a new form; enabling a better understanding of an emerging concept; encouraging the exchange of ideas, leading to greater peer review during the early design phase; enabling collaboration over a distance and at times when it would otherwise be impossible*" [Beardon, 1999c].

In a sense, this quotation is our guideline for this chapter and has been of central importance during the development of the *Visual Assistant* package[2]. To better support the exercise of design skills, it is necessary to better design software tools in general. This can be done through a relatively new discipline called 'software design'. But what is 'software design' ? What do we know about 'software' ? And 'design' ? We will try below to answer these questions before focusing on the creative practitioner and the user in general, and turning our attention to some design topics and concerns about software development that should help software 'practitioners' (engineers and designers) to allow an enhanced (without hindrances, centred on the user's experience) exercise of design skills with their applications.

We saw in the previous two chapters what is creativity and what is the relationship between creative practitioners and computers. In this chapter, we would also like to see how such studies of the context of practices – that will be supported by a future piece of software – can systematically be an integral part of the software development process that is concerned in producing usable quality software.

[1] See [Boden, 1992 ; Boden, 1994] for the study of such subject.
[2] See Chapter Four for an overview of the objectives and commands of the *Visual Assistant*.

An enlightenment on what 'software design' is

'Software design'. What is it ? Is it a new discipline or just another term for other disciplines also known as 'software engineering', 'interface design', 'human-computer interaction' ? How does it differ from activities such as programming or building software architectures ? Do we even know what 'software' and 'design' mean ? We should therefore take a closer look at each of these activities and ask ourselves how we could improve software by applying a broader understanding of design to our practices in software development – which are closer to engineering than designing techniques at the present time.

What is software ?

What is software ? Is it just the set of programs that you put into a computer when you want it to do particular jobs or something else ? Software is more than a set of implemented functionalities. [Winograd et al., 1996, p.xvi] put forward that software is "*a medium for the creation of virtualities – the world in which a user of the software perceives, acts, and responds to experiences*". In other words, software is like a (virtual) world, a space of existence in which the user lives, makes choices and acts. The way the user understands what the tool is and how that tool can be used, the way the interaction occurs, all that is adding to what the user is experiencing while interacting with the software. It is not only a set of images, devices and functionalities that are designed and, as a whole, make up what is called 'software'. Rather, it is something the user is experiencing, a space in which her/his actions can take place, within a context of values and needs that makes it possible for the user to understand and use the application. Software is not something "*we approach in isolation*" [Winograd et al., 1996, p.xxiii]. We bring with us our experiences – similar interaction experiences with other (piece of) software –, what we understand of the artefact (software is one among many of them), our expectations, the social/physical/historical context.

Software engineering

There is a misuse of the term 'software design' : generally, it is used to point out the discipline called 'software engineering'. But the centres of interest of the latter are not focused on the user's experience. Rather, it is "*the discipline concerned with the*

construction of software that is efficient, reliable, robust, and easy to maintain" [Winograd et al., 1996, p.xvii].

Computer scientists, programmers, software engineers, all are trained to the same principles for developing software. The former are bound to become scientists in a theoretical discipline while the latter are trained to focus almost exclusively on the construction of the internals of computer applications and, "*from the design point of view, give short shrift to consideration of use and users*" [Kapor, 1996, p.6]. The stance of the programmers is to implement the functional specifications without further questioning. In short, the main concern is the technical aspects, the mindset is 'problem-solving'.

All the practitioners mentioned above mostly follow the traditional method that consists of deciding what the system would do, then writing the specifications based on it, before implementing these specifications which includes figuring out how to produce interfaces – to which the user has to adjust. Usually, "*when software engineers or programmers say that a piece of software* works*, they typically mean that it is robust, is reliable, and meets its functional specification*" [Winograd et al., 1996, p.xvi]. About the latter, there is the misleading idea that "*the functionality of a piece of software is separable from, and takes precedence over, its appearance*" [Crampton Smith-Tabor, 1996, p.40].

Software engineers do not really centre their design on the people and their practices, the actions that the user will make in the real context of his work; however, they bring to the centre the technologies of computing. To put it in other words, "*software engineering has created an illusion that a rigorous process of transforming requirements* (stated in the specifications) *into systems is the key to reliable design*" [Denning-Dargan, 1996, p.107]. This goes hand in hand with a practice that doesn't consider, at the early stages of the development cycle, the "*appreciation of the overall conditions of use and user needs through a process of intelligent and conscious design*" [Kapor, 1996, p.4]. With software engineering methods, the designers are called (almost) at the end of the development process not to design the whole interaction, the user's experience. Instead, their action is confined to imagine approaches suggested by the way the system is engineered – more or less like patching with a Graphical User Interface (GUI) what the engineers have just implemented – but not by what users will do with the system (and the related information) and in which context of use.

What is design ?

Let us now consider what design is before looking how we could draw lessons, practices from this broad discipline and transpose them to the development of software. Such enlightenment – if put in practice – would help us to shift from the software engineering practices to the so-called software design discipline and would lead application builders to produce software that is more effective, more appropriate, and more satisfying for users – in brief, software that is better !

As [Denning-Dargan, 1996] points out, there are several meanings of the word 'design' that can be found in the dictionary. Let us mention a few of them. These definitions need to be complemented by the statements of professional practitioners to encompass all the dimensions, all the characteristics of design. However, and despite they are taken from common dictionaries, they tell us something about design.

> **Design :** "*Industrial aesthetics applied to the search for new forms fitting their function (for utilitarian objects, furniture, housing in general)*" [Grand Robert, 1993].
>
> **Design : 1)** "*The way that something has been planned and made, including its appearance, how it works, etc.*" **2)** "*The art or process of making a drawing of something to show how you will make it or what it will look like*" [Longman, 1995].

From the first definition, we can figure out that the industrial world has originally used design to 'wrap' its products, that is, to give some clues about its use, some meaning to the artefacts they mass-produce as well as to give them an appearance that makes them more attractive for the general public. The second definition reminds us of the primary meaning (the most frequent in dictionaries) of design which is 'to make or conceive a plan'. Both definitions mention notions such as 'how it works', 'what it will look like' but say nothing about the 'action/user-centred approach' – that is, keeping the user and her/his actions at the centre of design preoccupations –, the necessity for an artefact to fit the human body nor about the context of use, and many others.

First of all, one has to know that design is not only one discipline but a bunch of disciplines. Far from being exhaustive, we can list "*industrial design, graphic design, information design, urban design, and even fashion design*" [Winograd et al., 1996, p.xv].

In their activity of designing, these fields have in common many concerns as we shall see in the remainder of this chapter – the main one being the human experience.

Design is conscious. Design is not something that happens without we are aware of it. But that consciousness has some limits : you cannot say that, when designing, you are applying some theory of design. Design cannot be reduced to a set of rigorous, formal, consistent, or comprehensive rules unlike mathematics and engineering are. "*Systematic principles and methods at times may be applicable to the process of design, but there is no effective equivalent to the rationalised generative theories applied in mathematics and traditional engineering*" [Winograd et al., 1996, p.xx]. Thus, design is something you are aware of but which can be explained only in terms of feeling, intuition, tacit knowledge. Moreover, the designing activity is inherently unpredictable : you do not know where an idea will take you and how long it will take to get there – which is not the case for disciplines grounded in certainty such as mathematics.

Design keeps human concerns at the centre. To make it clear, we just need to interpret what a designer means when she/he says that something works. Any artefact that is well designed and, as a consequence, *really* works is an object that is well suited for its environment. Such a "*good design produces an object that works* ***for people*** *in a context of values and needs, to produce quality results and a satisfying experience*" [Winograd et al., 1996, p.xvi]. As it is emphasised by the words of the excerpt in bold, designers focus the design process on the quality, the richness of the user's experience.

Design is creative. And as a creative activity it tends to be messy and not reducible to a set of standard steps of a would-be method. Therefore, you can list any criterion for good design as possible, it will never turn someone into a good designer if that person has no skill of an artist-designer, cannot make use of creativity and cannot as well keep the richness of human experience at the foreground – pushing the computer and technical concerns in the background reducing them to supporting elements. This creativity appears in solving problems but goes beyond that activity typical of traditional engineering. But it also lies in "*finding the problems – envisioning the needs that people have but do not recognise*" [Winograd et al., 1996, p.xxii]. The marks of creativity that one can find in the activity of designing induces us to say that design – and many other practices based on it – is more an art than a science. And that artistic feature makes design difficult to learn through the kind of structured curriculum that is taught in faculties of science and

engineering. However, design is not a gift and education in the numerous design fields "*draws on the interaction between learner and teacher, designer and critic*" [Winograd et al., 1996, p.xxii]. As David Kelley (in [Kelley-Hartfield, 1996]) emphasises, besides keeping the human concerns in the centre, the designer must hold an important place for creativity and openness in her/his design practices. The degree of success for an artefact depends on the extent to which the designer is open to new possibilities and ready to take risks in a creative leap into those new possibilities that are radical novelties and whose consequences are not yet visible. To refer to Chapter Two, those new possibilities are like the parts of the conceptual spaces we had not noticed before or the limits of the same spaces. To take a leap into those new possibilities is like opening a door to an unvisited room in an existing house (exploring the unpreviously noticed parts of the conceptual spaces) or building a new house fundamentally different from the first (changing the limits of the conceptual spaces).

Design is communication. This communication is manifold. We know the user communicates, interacts with her/his environment and that the artist-designer also interacts with his/her materials (the medium of construction of every design activities). But the communication takes place as well between those two interactions. It has to be established between both of them to give all its meaning to the designed object, to guarantee that the object will be used by the user and well suited to its environment. At a surface level, the communication is twofold : an artefact conveys meaning about its content and about its use. This is done through tradition, learning and convention. "*A door communicates to its users through convention : a door with a flat plate near shoulder level says* 'Push Me!' *One with a round knob says* 'Twist Here!' " [Winograd et al., 1996, p.xxiii]. Communication is also peripheral – and at once less superficial. About designed objects communicating their meaning, you could make the analogy with the following proverb by saying that what one could see is only the visible part of the iceberg. In other words, as users we do not approach objects in isolation and therefore, someone cannot interpret the intended meaning without knowing the situational context. "*Every object appears in a context of expectations that is generated by the history of previous objects and experiences, and by the surroundings in the periphery – the physical, social, historical and working context in which the object is encountered*" [Winograd et al., 1996, p.xxiii]. As a result, something that has been newly designed inevitably takes meaning from what came before.

We mentioned about the 'problem-solving' mindset of software engineers. The mindset of artist-designers, at the opposite, is "*creating beyond what the problem calls for. (...) The designer (...) has a passion for doing something that fits somebody's needs, but that is not just a simple fix. The designer has a dream that goes beyond what exists, rather than fixing what exists. (...) The designer wants to create a solution that fits in a deeper situational or social sense*" and thereby goes beyond mere problem-solving [Kelley-Hartfield, 1996, p.153]. So, the designer's approach to a situation, to a problem is fundamentally different from that of an engineer : it is less problem-oriented but more open-ended. And this openness to explore (novel) possibilities also exists in the training/educational environment which makes it easier to try things and fosters the exploration and transformation of the space of expressive skills. Again, and more down-to-earth, "*design defines what* [object] *ought to be. By contrast, engineering* ***does it****. Engineering is implementation*" [Kelley-Hartfield, 1996, p.156].

What is software design ?

As we said above, software is more than a set of implemented functionalities – software is a world in which the user's experience takes place. Therefore, the development of applications shall move from software engineering techniques to a discipline called 'software design' which is mainly concerned with the quality of the above-mentioned experience for the people who use that software.

Has software design to do with interface design or human-computer interaction (HCI) ? Both are actually encompassed within software design which object of interest is not only software but also its surroundings. The overall design of a program is to be distinguished from the design of its user interface. Interface design is concerned about the interaction of users with computers through physical interfaces. All we experience while interacting with software applications is driven through the physical interfaces that makes the connection between the user and the machinery. Classical interfaces (keyboard, pointing device, visual display unit) and novel devices (virtual reality goggles and gloves, tactile input and output devices, more ergonomic keyboards, voice recognition devices, etc.) are creating and constraining at once the possibilities for user's experience offered by software. Therefore, the importance of interface design in the software development process is not insignificant. Human-computer interaction deals with "*the experience that people will have in encountering and using software*" [Winograd et al., 1996, p.xviii].

Human-computer interaction goes beyond the design of on-screen interfaces using visual standards, metaphors, icons with the same 'look & feel' – as it is usually taught in computer science curricula. "*Researchers in human-computer interaction have studied the mental worlds of computer users, developing approaches and methods for predicting properties of the interactions and for supporting the design of interfaces*" [Winograd et al., 1996, p.xviii]. However, as it is currently applied, human-computer interaction has not lead application builders to produce software that is more effective, more appropriate, and more satisfying for users. Human-computer interaction unfortunately takes place at the end of the software engineering process to wrap the application with a graphical user interface over the implemented functionalities.

Human-computer interaction or interface design do not make up software design but working in those fields of application development is already being engaged in that discipline at the opposite of software engineering mindset. That is what the Association for Software Design (ASD) claims. You even might be unaware of your participation as your payroll may refer to you as a programmer or a software engineer if not a human-factors consultant. But this does not give software design the status of an acknowledged profession – not yet; at best it is considered as a side task of a manager or a programmer. Still according to the Association for Software Design, we can give the following definition of software design :

> "**Software design** *sits at the crossroads of all the computer disciplines : hardware and software engineering, programming, human factors research, ergonomics. It is the study of the intersection of the human, machine and the various interfaces – physical, sensory, psychological – that connect them*" [Winograd et al., 1996, p.xv].

As the definition points out, software design does not dismiss these other disciplines that have often concentrated, in the education of computer professionals, on the understanding of computational mechanisms and on engineering methods that seek to ensure that the mechanisms behave as the programmer intends. It rather reconsiders the place of these disciplines in the software development process : all should be part of an overall, encompassing discipline (software design) that puts the richness of human experience forward whatever above-mentioned discipline is concerned. Software design is eventually a complex discipline as it draws on many disciplines : software engineering,

software architecture, programming but also human factors, graphic information design, art and aesthetics, sound production, psychology, and more. "*Software design, by the nature of what it aims to accomplish, sweeps into its scope all these interests, concerns, and disciplines*" [Winograd et al., 1996, p.297].

To make the preoccupations of software design clear, we believe that this definition needs to be completed by the accounts of professional practitioners engaged in software design from close or afar. In the words of [Kapor, 1996, p.4] – who was one of the first people in the microcomputer industry to identify his work as "*designing software*"; he was the designer of Lotus 1 - 2 - 3 spreadsheet – to be engaged in software design is to "*stand with a foot in two worlds : the world of technology and the world of people and human purposes*". Again, we are reminded of the central importance of human experience with software applications. This idea of mixing two different worlds is also perceptible in the view [Crampton Smith-Tabor, 1996, p.38] have of the software design enterprise : "*balancing a technological and an artistic perspective, with a focus on how people and designs communicate*". The artistic perspective is the one from the (artist-)designer that is in charge of shaping a piece of software in order to make it usable, effective.

> "***Software design*** *is concerned with the form and function of a software system and with the structure of the process that produces that system*". Software design, which is focusing on the action, is a discipline "*uniting system-oriented engineers and customer-oriented designers*" [Denning-Dargan, 1996, pp.107-108].

In this other attempt to give a definition of software design, we can spot again the balancing between two different worlds : the engineering discipline and the design discipline. Software engineering, which dates to the mid-sixties, is based in the engineering tradition, where design is seen as a formal process of defining specifications and deriving a system from them. Human-centred design is more recent, dating to the late eighties according to [Denning-Dargan, 1996]. In this approach of software development, designers immerse themselves in the everyday routines and concerns of their customers. Separately, these approaches do not lead to effective and usable software applications and their failing is in too much weaknesses. But put together, they build on their complementary strengths and form a broader approach of software development that we have called all along this chapter 'software design'.

"*Information technology is becoming an environment within which people operate rather than a device that they pick up and use. It is part of our everyday culture*" [Crampton Smith-Tabor, 1996, p.40]. To fully achieve the shaping of this new environment, the development of software applications needs to shift from engineering techniques to design techniques – that is where software design comes into play as it is focused on the human aspects rather than on the technical ones. Software design is like architecture for constructing buildings : you do not talk to an engineer first when you want your house to be designed and built, you go for an architect. It is not the job of the engineer – and she/he usually does not have the skills for that – to determine the design of your house according to the way a house is usually used and according to the particular use you will do of your house.

We already put forward the idea that software design cannot ignore scientific method and engineering knowledge. Indeed, familiarity with computing technology and psychology is as essential to a software designer as familiarity with building technology is to an architect. However, the fact that the rigorous, formal engineering, scientific techniques cannot be dismissed do not make the designing activity as formal and systematic as they are. "*Software design deals with values, preferences, and meanings – with, if you like, aesthetics and semantics – it can have neither a universal* predictive *theory nor always-reliable methods to generate solutions*" [Crampton Smith-Tabor, 1996, p.56]. Design is said to be more an art than a science. Though software design is one of the numerous fields of design and, subsequently, a little unpredictable, it is neither a science, nor an art : it combines both aspects.

Software designers and 'product' designers

Before going further with the development of this chapter, we need to make clearer the distinction between two design traditions and their practitioners. According to [Beardon, 1999b], 'software' designers and 'product' designers are different words depicting different concepts. The former is designing software technology for individuals to use. Some of these individuals are for instance standard office applications users or ICT professionals while the others are product designers. Product designers therefore use the software technology developed by the software designers to devise (actual or virtual) products. Their role in this context is the one of the user – they use the applications developed by software designers. They are the creative practitioners mentioned all through

the previous and following chapters. And their creative practices require suitable software design methods otherwise they will feel hindered in their search for the unexpected behaviour of the packages they use.

Software engineers prefer to consider application design from their "*isolated world in which they are locked*" [Beardon, 1999b] : they involve the user in the development process mainly because 'they have to' and they seem – so far –to be very less sensitive to the needs of creative practitioners than software designers are or could be. They confine themselves to traditional software engineering methods as they find easier to implement metaphors including what they think to be current user practice.

As far as the creative practitioners are concerned, on the one hand they refuse to "*engage in constructive criticism*" [Beardon, 1999b] of the tools they use. It is as if the tool fits their usage – so they keep using it – or not – in this case, they get rid of it. On the other hand, many artists and designers proceed by exploring the unexpected behaviours of applications in order to get interesting results and base their design works on these results. Doing so – refusing to engage in critical use of software that they consider just as tools; forcing the software to misbehave, subverting the intentions of software designers – they contribute to enlarge the rift between software and product designers.

Ways to improve the development of software

In the previous sections, we saw the flaws of software engineering methods. We should therefore run away from the conventional wisdom on software engineering. But before, let us have a brief look on the development process in software engineering and some of its models to better understand and review the development of software. Then, we will consider how we could improve the usability, reliability, dependability, appropriateness and rely better on the quality of the user's experience.

Traditional software engineering cycles

Software engineering is a discipline originating from the mid-sixties. Software engineers believe that most of their work lies in the process that generates a system having the form and function specified by the customer. They refer to this as the 'software-lifecycle model'. This model declines itself in many varieties of which the most common

are the 'waterfall model' and the 'spiral model' – also called the 'prototyping approach' or the 'iterative model'. Many authors have given detailed accounts of how these process are organised. Here, we just give a brief explanation of the most common models and principles. For more information, we can refer to the work of [Somerville, 19??], Andriole and Freeman, [Boehm, 1988], DeGrace and Hulet-Stahl, and Dijkstra or at any text book used for software engineering lectures.

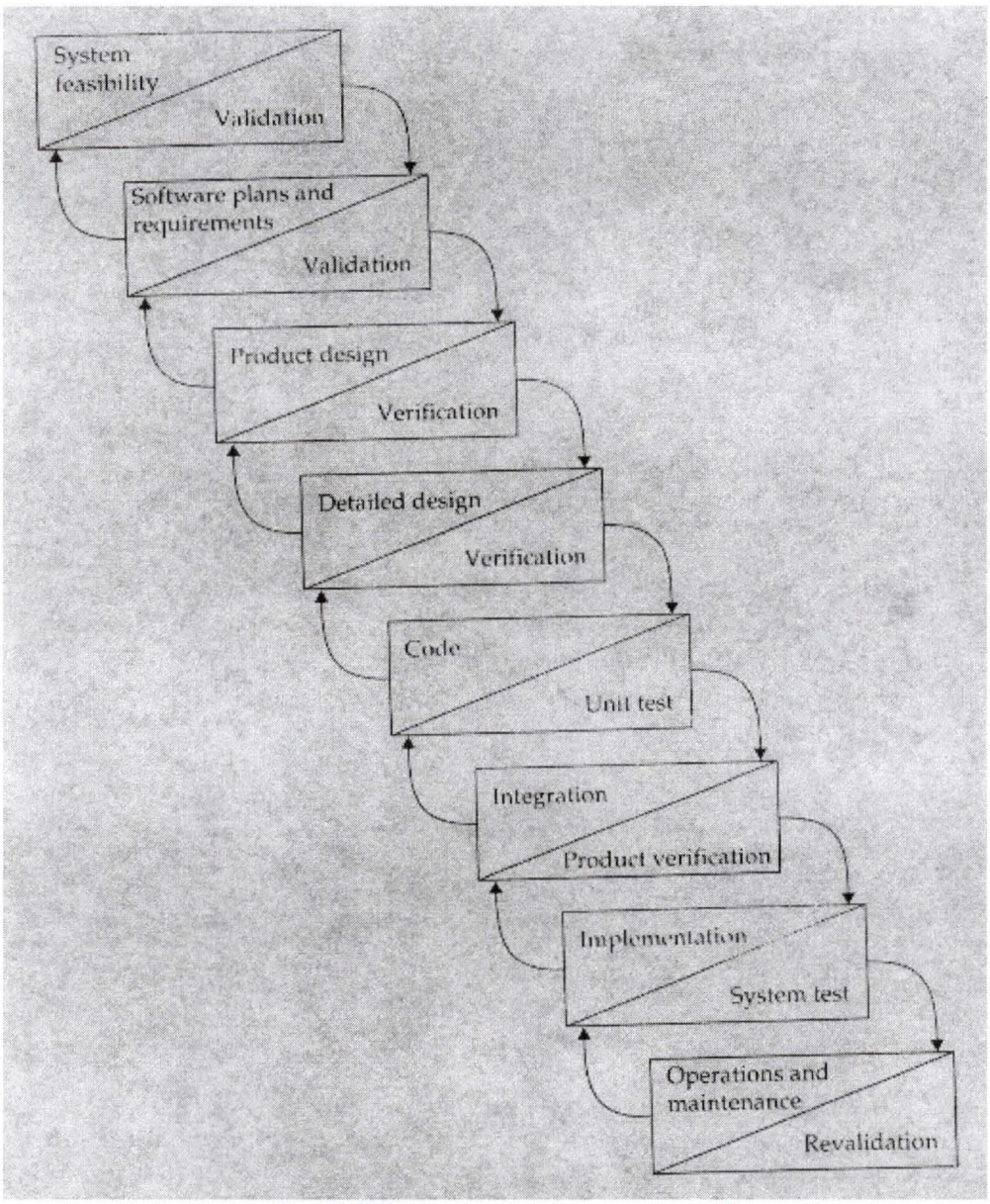


Fig. 3.1 : The traditional waterfall model of the software engineering process.

The traditional waterfall model of software development (fig.3.1) emphasises the structured relationships between adjacent stages in an idealised process that moves from abstraction to implementation. It is idealised as the detailed design, and sometimes the

product design, are either actually addressed only after the code has been developed or constrained by the requirements put into specifications by the software engineers. Moreover, it does not take into account the realities of iterative design[3], such as the prototyping cycle.

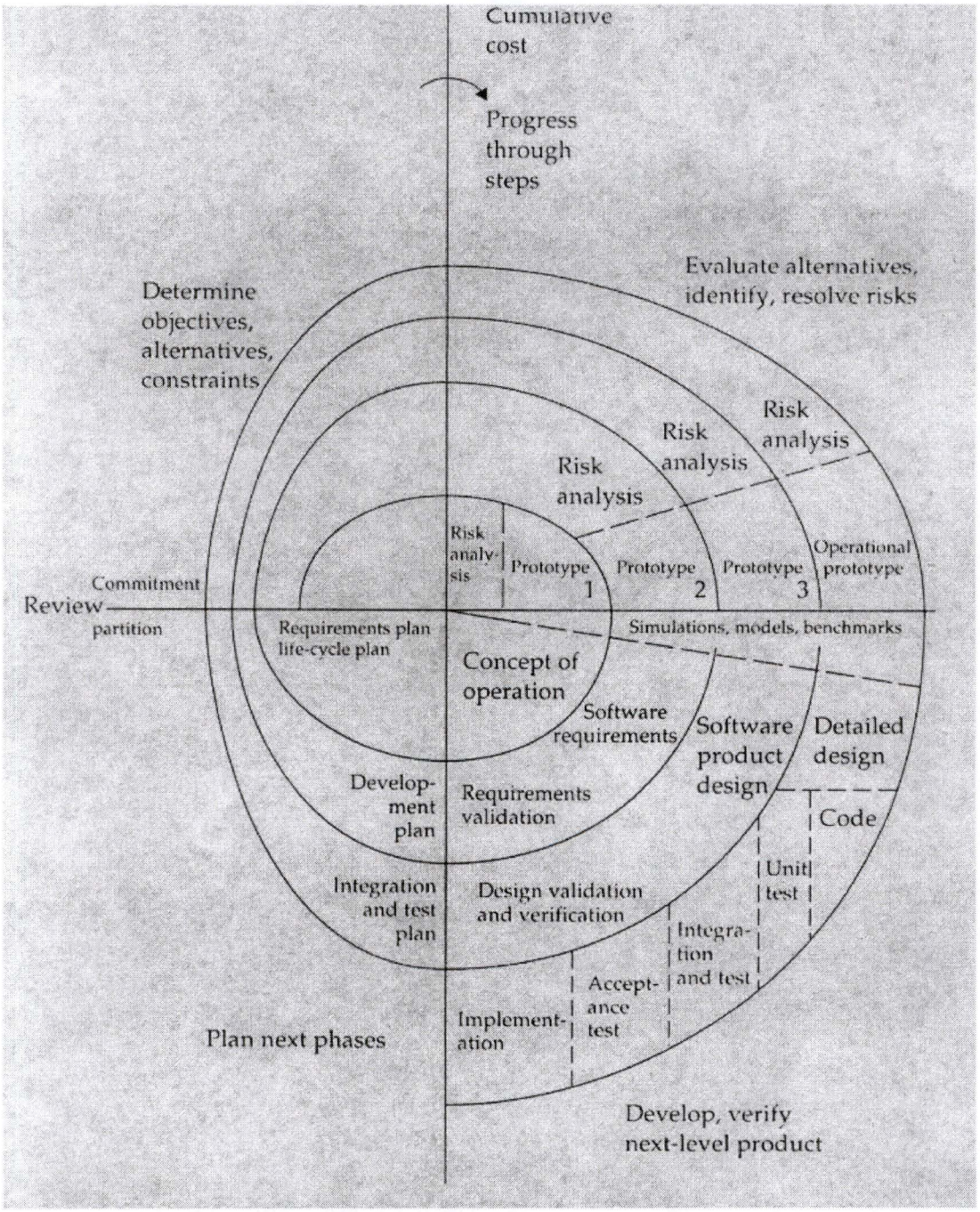


Fig. 3.2 : the traditional spiral model of the software engineering process.

The spiral model (fig.3.2) improves on the waterfall model described above by emphasising the iterative nature of design process. It introduces a cycle of iterative

[3] See the following sections.

prototyping, but it is still functionality and product centred, rather than user and action centred. Although the design process and its iterative nature are more discernible in this model, the people in charge of the design are not often design practitioners. This designing job is most of the time left to the programmers or engineers as a side task – task to which they give short shrift according to its nature. Again, the design, detailed or not, is addressed too late in the process – confining designers to imagine approaches suggested by the way the system is engineered.

A shift in view

There is an analogy that makes the problem with software engineering more striking. Suppose you have decided not to buy a house but to ask a company to build it. In this case, you go to see first an architect, then the building engineers. In software engineering, the process is inverted : the software engineers establish the software requirements in collaboration with the managers of the client company without observing users' behaviour and context of work or world of action; and then, only later in the process, the designers are called to lay a Graphical User Interface (GUI) over the implemented functionalities. In the following sections, we will try to reverse that course of things and give a few guidelines to improve the conventional methods. These reviews are based on the practices that are prevailing in design activities such as keeping human concerns in the centre, observing user's behaviour in their real context of work and action, etc.

The inverted process in software engineering operates from three assumptions, according to the observations made by [Denning-Dargan, 1996].

1. The result of the design is a product (artefact, machine or system);

2. The product is derived from specifications given by the customer. In principle, with enough knowledge and computing power, this derivation could be mechanised;

3. Once the customer and the software engineer have agreed on the specifications, there is little need for contact between them until delivery.

This methodology, because of its standpoint, has lead to software applications that are not so usable and dependable. That explains why the software crisis is usually seen as a breakdown in the application of this methodology. But when one says that it is "*especially due to faulty input in the form of incomplete or incorrect specifications*", as reported in [Denning-Dargan, 1996, p.108], we do not agree. The problem is more fundamental than that : it is the software engineers doing (badly and carelessly) the job of designers, the lack of observation of the users, and engineering practices at the opposite of design practices. Facing these shortcomings, some emphasise the role of human judgement, experience in work, interaction and coordination among people and organisations and argue that the software crisis is fundamentally a failure of customer satisfaction.

"*The standard engineering design process offers little to connect the actions of designers with the concerns of users; the interaction between these two groups is limited to the requirements and specifications documents and to the final signoff*" [Denning-Dargan, 1996, p.111]. The product-centred design process focuses primarily on the machine and its efficiency, expecting humans to adapt. In contrast, as Donald Norman says – quoted by [Denning-Dargan, 1996, p.111], "*the human-centred design process leaves to humans the actions that humans do well, such as empathising, perceiving, understanding, solving problems, listening for concerns, fulfilling commitments, satisfying other people and serving interests. It leaves to machines what humans do not do well, such as performing repetitive actions without error, searching large data sets, and carrying out accurate calculations*".

Human-centred design, better known as software design, is based on understanding the domain of work or play – the artistic domain of action for theatre scene designers – in which people are engaged and in which they interact with computers. In addition to this, software design is also based on programming computers to facilitate human action. As we did for software engineering above, here is the three assumptions on which the software design methodology is grounded – based on the work of [Denning-Dargan, 1996].

1. The result of a good design is a satisfied customer;

2. The process of design is a collaboration between designers and customers. The design evolves and adapts to their changing concerns, and the process produces a specification as an important "*by-product*";

3. The customer and designer are in constant communication during the entire process.

The comparison of both standpoints has lead us to think that software engineers must shift software development from a process of deriving a software system from specifications documents to a process of supporting the standard practices in a domain through software that enables more effective action.

Core activities of any design process

In their study of the interactions between people and products of any kind – including software – [Crampton Smith-Tabor, 1996] have determined five core activities of any design process. These activities, given hereafter, can also be found in the software design process, balancing technological and artistic (aesthetic and human-centred) perspectives. In the following sections, references to these five core activities of design will be made to emphasise on the beneficial influence of design on software design.

1. *Understanding.* 'What is going on here ? What is the underlying problem to be solved ?', could you ask. Photographs, videos, sketches, and notes can be used to aid designers in observing and analysing the information or the problem. David Liddle and his software design team working on the development of the Xerox's Star (the first system having a Graphical User Interface) used 600 to 700 hours of such video recordings filming people interacting with computers and with prototypes of the Star [Liddle, 1996]. Besides, designers talk with people, especially clients and users, and look at the information to be communicated.

2. *Abstracting.* What are the main elements ? What kind of information is being conveyed ? What do people want to do with it ? What is important ? What is irrelevant ? Lists, sketches and diagrams are the usual tools here.

3. *Structuring.* What are the relationships among the elements ? What different ways can the elements be ordered to be useful

for users ? What are the users interested in ? How much can they take in ? The designers' assumptions will be checked with the users and the clients.

4. *Representing.* How can this structure be represented in visual and auditory form ? What representations does the material suggest ? What representations might the designer glean from thinking about the users' world ? Should the representation be concrete or abstract ? Is metaphor appropriate ? Here, the designer typically uses sketches on paper and interactive sketches in a software medium, which may be evaluated with colleagues or users.

5. *Detailing.* Exactly what colour should this element be ? What style of depiction should be used ? How is the picture plane handled ? How do the elements move ? Should an illustrator be hired ? Some designers work directly in a paint program. Other start on paper, and move to the computer later.

These five processes are not executed sequentially. Designers' skills lie in circling among them as activity in one throws light on another. And, one more time, the stage at which the designers are admitted to the design process determines the extent to which the full range of their skills can contribute to a (software) product's success.

Approach to software design[4]

Task analysis

Software systems have customers. Quality means customer satisfaction. This statement is well known by people working on the development of quality standards. But what does software quality consist of ? Customers are more likely to be satisfied by software that is 'transparent' in their domain of work. 'Transparent' means that the computer slides to the background, allowing the users to perform familiar actions without distraction, and to perform new actions that previously they could only imagine. From this, we conclude – as

[4] This approach is based on the work of [Denning-Dargan, 1996] and [Liddle, 1996].

[Denning-Dargan, 1996] and many others did before us – to the necessity to observe the behaviour of the user in its context (See 1. Understanding). [Liddle, 1996], who headed the development of the Xerox's Star System (the forerunner of today's graphical user interfaces (GUI's), emphasises this necessity by saying that the development team for the Star System "*developed [its] design principles from what [they] saw users doing*".

Do not think of the software design's task analysis in conventional terms. Our perspective, inspired by the account of [Denning-Dargan, 1996], is a distinct departure from the traditional functional analysis of a domain, which describes the domain only as a process network – a network of interconnected input-output functions that operate on physical or information objects. We understand task analysis here in the sense of studying the nature of the actions that people take in their domain, especially of repetitive actions – they are the kind of actions that computers do well, to connect those action-processes to supportive software technology. And all these actions are related to speech acts indicating when the tasks are declared, initiated, or completed and to which degree of satisfaction. What do they complain about most ? What new actions would they like to perform next ? Which actions are endlessly repeated by them ? What is the context in which their actions take place ? All these questions find answers in the observation of user's behaviour in context – and not by deciding in the isolation of the project leader's office what will be the tasks to be performed in interaction with the computer software.

Software design practitioners must be – or must become through adequate training – skilled observers of the domain of action in which a particular community of people engage, they must know that to observe the repetitive processes, they have to watch what people say to one another (verbally, in writing, via body language or e-mails) so that the designers can produce software that assists people in performing those actions more effectively. The phrase 'domain of action' is meant to be broad; it includes specialised domains, such as "*medicine, law, banking, and sports*", as well as general domains, such as "*work, career, entertainment, education, family, health, world, dignity and spirituality*" [Denning-Dargan, 1996, p.112]. Again, [Liddle, 1996, p.28] reminds us the importance for software designers to immerse themselves in the user's domain of action when he says that "*designers make their greatest user-interface errors when they don't think about users in terms of what those users are doing in their jobs*".

As [Denning-Dargan, 1996] point out, the assessment of whether software is useful, reliable and dependable is made by the people who act in a domain. By focusing on a system and its specifications, the conventional software engineering process loses sight of those people, of their common concerns and of their actions as they do not take time to observe them performing some tasks in the context of their work or any other context. One has to know that, in this particular case, the process cannot offer a grounded assessment of quality, because many of the factors influencing quality are not observable in the software itself. Once the user concerns have been determined, the next stage consists of making connections between these and the structure of software. But this cannot be done systematically and requires first to build a user's conceptual model / ontology of the domain and then to iterate through a prototyping cycle.

Building the user's conceptual model / ontology of the domain

The user's conceptual model represents, in [Liddle, 1996, p.21]'s words, "*what the user is likely to think, and how the user is likely to respond*". The use of computer metaphors (e.g. the desktop or spreadsheet metaphor) is not intended to imitate office furniture or other physical objects. Their purpose is to let people use recognition rather than recall. People are good at recognition, but tend to be poor at recall. People can see objects and operations on the screen, and can manage them quite well. But when you ask people to remember what string to type or what actions to do to perform some task, you are counting on one of their weakest abilities. So, the use of computer metaphors can help to guide (and predict) what the user is likely to think. In the end, the effectiveness of a piece of software, and of its interface, depends on what they think about it, on how they are interpreted by the users. Furthermore, these computer metaphors have the critically important role to be abstractions that users can then relate to their jobs to help them taking (repetitive) actions. This second role, through the development of scenarios for uses of the imagined product based on the task, helps to determine what the user is likely to respond.

The mental model described by [McCullough, 1998] is similar to the user's conceptual model above. Malcolm McCullough places his talk upstream by naming the weakness of people 'rote knowledge' rather than 'recall', in the terms of David Liddle. A rote knowledge of procedures does not necessarily include an understanding of underlying system states of a computer. He says that effective mental models of computer system states – in other words, what the user thinks about the computer and what be done with it –

are able to predict outcomes of untried procedures and are essential for learning how to combine elements of the digital repertoire.

What David Liddle called the 'user's conceptual model' has been named the 'ontology of the domain' by Peter Denning and Pamela Dargan in [Denning-Dargan, 1996, p.117]. An ontology, according to them, is "*a conceptual framework for interpreting the world in terms of recurrent actions*" (See 2. Abstracting). It is a sort of study of the context of use and its purpose is to use it to coordinate the work of the software developers during the next stages of the development process. The software designer building this ontology of the domain and using it to guide the software engineers is analogous to the architect creating sketches and blueprints and using them to coordinate builders as well as to help customers assess the results.

This brings us to the question of the features we would expect from such an ontology of the domain. According to [Denning-Dargan, 1996], an ontology should provide three elements :

- It should convey the patterns of action of the domain in which the software will be used, in terms of its basic distinctions, repetitive processes, standards of assessment, strategies, tools, breakdowns, and driving concerns;
- It should connect the linguistic structure of the domain to the software structures that will support the patterns (See 3. Structuring), and to guide software engineers in implementing those structures;
- It should provide a basis for measuring the effectiveness of the implemented system in practice.

This ontology represents the domain of action rather than just the system being built (its standard elements of program specifications or computer code – which are produced later in the software design development process). We listed below the patterns of action of the domain :

- A set of *linguistic distinctions* (verbs, nouns, jargon, etc.), around which people in the domain organise their actions

➔ in the domain of theatre, these distinctions include play, stages, stage sets, props, blocking[5], actors, stage set design, rehearsals, producer, the lines of actors, act drop, the audience, etc.;

- A set of *speech acts* by which domain participants declare and report states of affairs, initiate actions, signify completions, and coordinate with other people

  ➔ in the domain of theatre, these speech acts include 'acting!', design the stages, place a prop, change the settings, start the rehearsals, play opposite an actor, repeat your lines, drop the curtains, etc.;

- A set of *standard practices* (recurrent actions, organisational processes, roles, standards of assessment) performed by members of the domain

  ➔ in the domain of theatre, these standard practices include making rehearsals before the performances, working out the place of actors, choosing the props, designing the overall look of the play, sketching the stage settings, casting the parts, etc.);

- A set of ready-to-hand *tools and equipment* that people use to perform actions; a tool is ready to hand if the person using it does so with skill and without conscious thought

  ➔ in the domain of theatre, these ready-to-hand tools include props, performance dresses, scripts, actor's lines, lighting system, backstage, etc.);

- A set of *breakdowns*, which are interruptions of standard practices and progress caused by tools breaking, people failing to complete agreements, external circumstances and so on

  ➔ in the domain of theatre, these breakdowns include broken lighting system, ripped dresses, actors not knowing their lines,

[5] The process of working out where the actors will be standing during particular parts of the play.

wobbly chairs and rickety tables, stuck curtains, missing colour filters, etc.);

- A set of *ongoing concerns* of the people in the domain : common missions, interests and fears

  ➔ in the domain of theatre, these concerns include rapid stage sketching of a play being designed, availability of props for rehearsals and performances, actors not falling sick, to do the advertising for the play, new dresses sewed in time, etc.).

This ontology will serve during the following stages to keep the context of use in the centre of the development concerns. It will give the people in charge of prototyping (the next phase of the development process) a better knowledge of the domain of action and will help to keep the focus on the practices in use in the domain. It is like a pre-version of the specifications on which the prototypers will base their work (the development of the interfaces and interaction patterns constituting the prototypes). The test users of the prototypes will use this ontology as a landmark to assess the prototypes derived from it.

Prototyping cycle

Customer satisfaction is not static as they can change expectations during the development process as well as after completion of that process. Consequently, software must evolve to track their shifting expectations respectively by going through a prototyping cycle (See 4. Representing) and by releasing new versions of the software packages. The study of the user's reactions does not stop after the task analysis but goes on through the prototyping process.

The deployment of prototypes early in the software development process – preceding any writing of code ! (See 5. Detailing) – can help to observe how people react and what kinds of breakdowns they experience. Because the customers frequently shift concerns, the software designers must have means of observing shifting concerns and of taking these shifts into account in the next version of the design. Prototypes are the perfect means for that observation but they need to be backed by techniques like "*test sites, individual follow-up sessions, hot lines, highly attentive technical and customer support staff,*

suggestion boxes, bug advisories, and the like" [Denning-Dargan, 1996, p.114-115] as it is of central importance to stay in communication with customers.

During this time laps, there is the great temptation for software developers – having the state of mind of an engineer – to build prototypes designed around what would run fast and be small and easy to implement. To focus on taking advantage of the specific power of the platform is a standard practice in the conventional software engineering development process. To eventually produce a software product that is usable and enhances the user's experience logically requires to keep concentrating on the user during the prototyping cycle – and all along the development process as well. A quotation of [Liddle, 1996, p.25] illustrates our words : "*Our attitude was always 'Wait a minute, let's make this work for the user. If we find we can't implement something, then we'll go back to the drawing board. But we're not going to pick things that we think will be small or fast to build, and then bully the user into accepting them'* ".

The implementation must not come in the first place : if the user's says that it is okay with the design of the prototype, only then the development process can evolve a bit further. David Liddle and his team at Xerox wrote a 400-page functional specification before they ever wrote one line of code. But they did not just sit down and write it. They prototyped a little bit, did some testing with users to decide what made sense, and then wrote a specification for that aspect. Then they prototyped a bit more, tested it, and then specified it again, over and over until the process was done [Liddle, 1996].

Adapted computer science curricula

Everyone would agree that the education of computer professionals has often concentrated on the understanding of computational mechanisms, and on engineering methods that, in accordance with a formal and rigorous process, transform specifications describing the software functionalities into their implementation (the derived system). Over and above, the focus is on the objects being designed : the hardware and software. The primary concern is to implement a specified functionality efficiently. Reliability, robustness, conformity with the functional specifications are important in software engineering. Indeed, any developer who ignores them could see her/his application crash or misbehave which can be highly dangerous in particular contexts of work.

But this perspective, with its focus on function and construction, is like wearing blinkers not to see that the applications developed are most of the time not meeting the needs of people, are not working in the context of work or user's world of action, do not produce quality results nor a satisfying human, user's experience. To strive against the lack of usability of software and the poor design of programs, computer professionals themselves should take responsibility for creating a positive user's experience. "*Perhaps the most important conceptual move to be taken is to recognise the critical role of design, as a counterpart to programming, in the creation of computer artefacts*" [Kapor, 1996, p.3]. A development process in which design is a central piece is what we described and called 'software design' all along this chapter.

Since the early nineties Mitchell Kapor made the case that we need to think software design as a profession, rather than as a side task of a manager or a programmer. And the best way to achieve this is in introducing design activities in the curriculum of computer science. Such an adapted computer science curriculum should substitute software design for software engineering lectures – dealing with traditional software engineering development methods. This should be a challenge for the next years for the universities and high schools proposing computer related curricula – at the moment, very few universities – such as the Stanford University – have developed programs in software design.

Putting up a strong case for moving from an engineer's-eye view to a designer's-eye view and for taking the system, the users, and the context all together as a starting point for a curriculum in software design is easy to say. But what expertise do software designers need, then ? According to [Winograd et al., 1996, p.297], "*they need to be able to envision, to create, and to develop a representation of their vision that they can communicate. They need to be able to speak the language of all the people involved in the enterprise addressed by software design : the user, the programmer, the graphic designer, the database architect, the marketing specialists, and all the rest. They need to be able to understand each discipline well enough to know when to involve relevant collaborators and how to incorporate the contributions of experts from other disciplines into the software design visions that they create*".

"*Naturally, programmers quickly lose respect for people who fail to understand fundamental technical issues*" [Winograd et al., 1996]. The answer to this is not exclude

designers from the process, but to make sure that they have a sound mastery of technical fundamentals, a firm grounding in technology. This would give them the assurance that they are an effective participant in the overall process and speak the language of computer practitioners. A curriculum in computer science and software design should propose to the designer student courses that deal with the principles and methods of computer program construction. Topics, in Terry wynograd's view, would include computer systems architecture, microprocessor architectures, operating systems, network communications, data structures and algorithms, databases, distributed computing, programming environments, and object-oriented development methodologies. Software designers must have a solid knowledge of at least one modern programming language in addition to exposure to a wide variety of languages and tools.

Conclusion

After having answered to the initial questions of 'What is design?', 'What is software design?' among others, we saw in this chapter the importance of the stage at which the designers are admitted in the process, the importance to listen to the practitioners of a domain and effectively take into consideration the context of action in which the software application will take place to support their practices. The reader shall have noticed that the previous chapters were dedicated to understand what creativity is and the domain of action of artists and designers. Such work, in an even more detailed form, should be conducted as part of the software development process. Applying the software design principles to the development of applications aimed at supporting the creative practices should lead to effectively used pieces of software, contributing to enhance the user's experience and to reduce the hindrances as well as it should open the software developers' eyes to the existing needs and, in particular, the need for sketching tools, not only in the domain of theatre – which is the subject of the Visual Assistant and of my degree training – but in many other artistic domains as well.

MICHE Frédéric
Master's Degree of Computer Science,
5th Year

Master's degree dissertation

Chapter Four

Visual Assistant : A support for creative practices for theatre

Main advisor : Jacques BERLEUR, s.j. (FUNDP) Training advisor : Colin BEARDON (ESAD)

Chapter Four

Visual Assistant : support for creative practices in the theatre

Introduction

To illustrate our study let us have a look at the *Visual Assistant*. This software exists in two versions : one developed for the Apple Macintosh (version 1.3 at the time of writing) and one developed for the PC. As the former is relatively mature, we have to mention that the latter is not fully implemented – thus, we called it *VA prototype*. The author of the Macintosh version is Colin Beardon, the 'Training Advisor' for this master's degree dissertation. The PC prototype has been developed by myself with his advice as part of my master's degree dissertation. Other students in computer science or professionals (designers or computer scientists) may have to further develop the PC version of the *Visual Assistant*.

In this chapter the objectives and an overview of the commands – as implemented in the Macintosh version – of the *Visual Assistant* (also called *'VA'* for short) are first described, then followed by what the prototype running on PC's actually implements and what is still to be done to fulfil the objectives. To close the chapter, a user guide for the PC prototype is given.

Objectives of the *Visual Assistant* project

The *Visual Assistant* project, [Beardon, 1999a], develops software to enable people working in the theatre – and concerned with performance – to express and communicate their visual and spatial ideas through the use of digital technology. It is especially concerned with the interplay of such software with the creative and working practices related to performance.

[Beardon, 1999c] says that: "*It can be argued that software already exists for such a purpose in the form of text processing, 2-D image processing, 3-D model building and multimedia applications. As the outputs from such applications are increasingly exchangeable across hardware platforms, why is there a need for another software package ? We suggest three potential answers to this question :*

- *during the early, 'sketching' stage of a design it is necessary to work with ill-defined ideas and existing packages require too much detail;*
- *2-D graphics does not enable the proper exploration of three dimensional space, whereas 3-D modelling requires technical skills that are not easily achieved;*
- *and none of the interfaces look and feel like a theatre – they look and feel like a computer interface".*

Initially, ideas come in a rough, non-detailed form like "*impressions and intuitions*" [Beardon, 1998]. An artist or a designer, during the initial stages of creation when concepts are in formation, will try to capture these on paper. Because intuitions do not last very long, there is a critical need to rapidly sketch different representations of these until they manage to get as close as possible to what they had in mind. As a consequence, many sketches will then be discarded.

This should help us – as software designers – to better catch the crucial need for sketching tools when there are so many packages concerned about detailed representations. [Beardon, 1998] points out that those packages could be "*positively harmful (...) : a 3D computer modelling system that requires two or three days to build a model of a set can be very impressive, but is not appropriate for this early stage of design. We simply cannot afford to spend days on a design, only to find that it does not capture our original idea. The tendency, having invested so much time, is to stick with the design and try to make it better, whereas what should be done is to discard it and try again*". We do believe that "*when not hindered by a need for accomplished draughtsmanship* or photo-realistic quality

the visual process becomes more accessible[1]" [Beardon, 1999c], thus can be used in teaching context.

"*The software was designed according to what is stated above and for users who are primarily interested in the theatre and are probably not at all interested in computers. Many years' experience of working with such students led* Colin Beardon (the software designer and developer for the original version of the Visual Assistant on the Macintosh platform) *to some initial design objectives for the software* [Beardon, 1999b] *:*

- ***It must be very simple to learn*** *: you should be able to see someone else using it and then confidently use it yourself. No manual, no tutorial !*

- ***It must be simple to use*** *: there is little point in providing 96 options when most users will only use 12.*

- ***It should give meaningful results quickly*** *: there is a tightly iterative process of producing and evaluating.*

- ***What happens should be like sketching*** *: it should not matter if work is irretrievably transformed or destroyed — you can always try again.* The Visual Assistant should rather be seen *as a theatre image-making sketch book rather than as a theatre design tool.* As explained in Chapter 2, the *Visual Assistant* has been developed as a sketching tool to offer artists and designers tools that can be used during the initial stages of the creation process – which does not need details before its final stages; Chapter 2, Concept formation and sketching.

- ***It should support 'process' rather than 'product'*** *: particularly processes that leads to clearer understanding and better actions in the real world,* e.g. a better perception of a play to put on.

- ***It should present a believable 'language-game'***[2] *: when acting on the computer screen you should be thinking as a theatre person –*

[1] This quality is part of the ones required by the creative practitioners and listed in the typology. See Chapter Two, Typology for that typology.

that is, the software-tool should moved aside, hide behind the theatre practitioner. It should become 'transparent' for its user rather than constrain him or her to submit to its 'policies[3]'.

- ***It should support person-to-person dialogue*** *: it should function as a common sketchpad to support critical discussions.*

- ***It should be able to lead to more detailed implementation***".

These design objectives should make it easier for the theatre practitioners using this sketching tool (the Visual Assistant) to explore unpreviously noticed parts or the limits of the conceptual spaces[4] related to creative practices and to possibly tweak and transform (the two sorts of changes distinguished by Margaret A. Boden and described in Chapter One) them, which can lead to creative or radically creative novel ideas for the design of plays.

The *Visual Assistant* software enables the rapid prototyping of visualisations without the need to cope with full 3D computing. These may be visualisations of the physical stage, but they may just as profitably be visualisations of other aspects of performance – a.o. "*to give a multimedia presentation of the HaMLET project, analyse the blocking[5] or as a means of analysing a play text*" [Beardon, 1998]. Experience has shown that many users adopt a looser approach to what is being represented. Some of benefits claimed by users include : an ability "to create 'atmosphere' rather than detail"; an ability "to describe an environment within which action can take place"; and "a medium for visual improvisation[6]".

The package (the Macintosh version, as it is – fig. 4.1) is easy to learn and use, and produces output that can be viewed locally or transferred over internet as VRML files. These can then be viewed on any platform using a standard 3D-browser plug-in (fig. 4.2).

[2] Such languages are used to design objects that express what the objects are, what they do, how they are to be used, and how they contribute to experience. They are the visual and functional language of communication with the people who use an artefact [Rheinfrank-Evenson, 1996].
[3] Understand it as its way of proceeding.
[4] For an explanation of the 'conceptual spaces', see Chapter 1, Conceptual spaces.
[5] The process of working out where the actors will be standing during particular parts of the play.
[6] Understand it as 'do not have pre-conceived ideas, do not let technology or material issues guide your actions. Feel free to explore new ways, new visual ideas, new designs and go beyond coping with technology'.

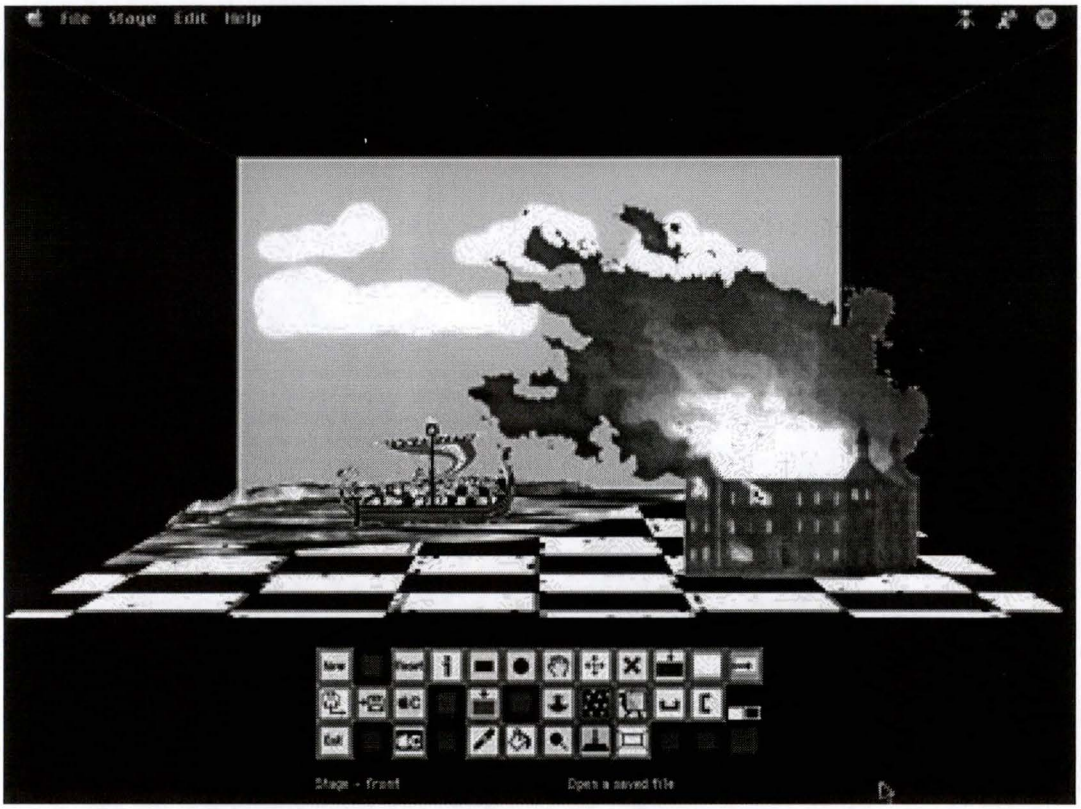


Fig. 4.1 : Designing a stage set with the Visual Assistant on the Macintosh platform.

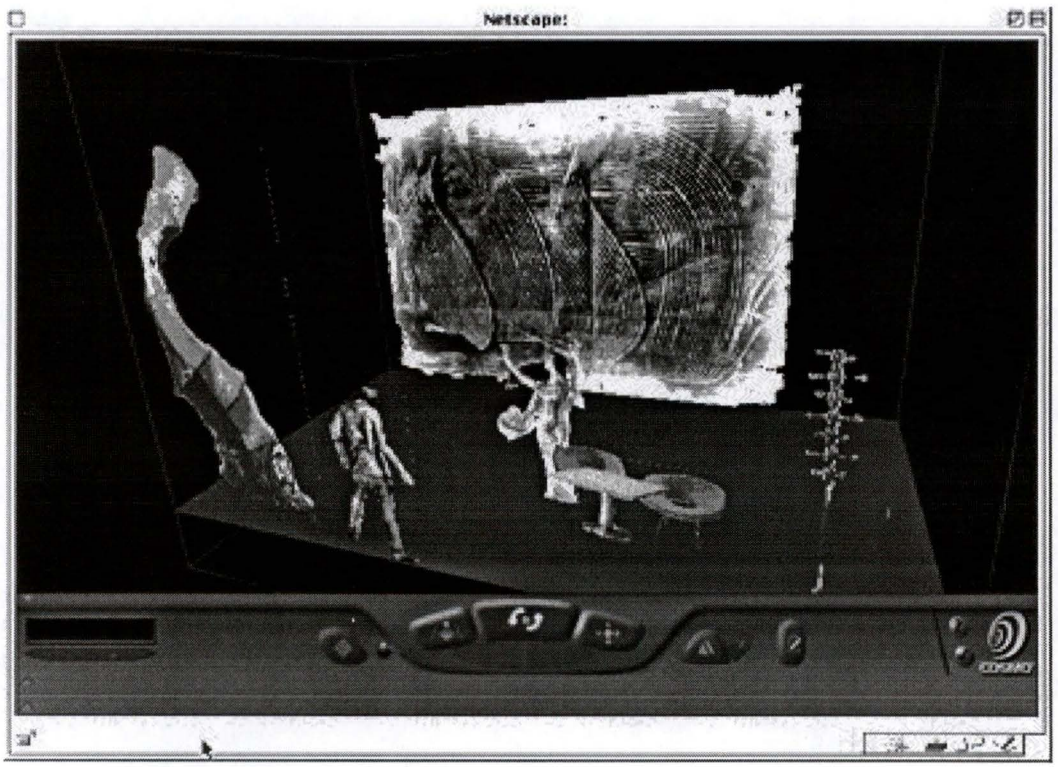


Fig. 4.2 : Viewing a stage saved on the Macintosh platform with the VRML file format.

Most of the work of Margaret A. Boden was concerned with trying to embody creativity in a computer programme. As one objective in Artificial Intelligence is to reproduce human behaviour inside computer machinery, her objective lies in trying to give a computer creative drives, to make them produce some works of art. This is not the way of looking at things underlying the concepts that the *Visual Assistant* package implements. In this project, we are rather interested in supporting human creative practices by using software applications mainly for the early stages of the creative process.

In Chapter Two, we listed many concerns regarding the needs and reproaches of artists and designers in terms of software applications supporting their creative practices. In what follows, we look at the concerns that are addressed by the Visual Assistant. The concerns, as given in Chapter Two, are formatted in bold; the answer given by the Visual Assistant is placed just below, following an arrow.

In the relation between artists/designers and computers, there should be...

- **A feeling of greater independence in the choice of tools;**

  ➔ the Visual Assistant is made for people working freelance – wishing to visualise the play they are writing – as well as for people working for a company producing plays already written – wishing to determine the best visual aspects for the play by quickly comparing different designs, different ideas.

- **Suitable software tools to use during the concept formation. These tools should offer the possibility to sketch, and consequently to explore, ideas rather then requiring details right from the start;**

  ➔ the Visual Assistant is a sketching tool to develop and compare ideas regarding the design of the different stages of a play. It is especially designed for use during the concept formation. The Visual Assistant does not render the overall look of a stage with a particularly photo-realistic quality as this feature is not desired in the early stage of design.

- **Simplicity is often lacking in many software applications. There is a preference for *"packages that present themselves initially as quite simple, but reveal depths of complexity as you get to used them"*;**

  ➔ the Visual Assistant offers simplicity in moving objects back and forth rather than erasing pencil lines and redrawing objects backwards or forwards on a sheet of paper.

To use computers, artists and designers need…

- **A sense of good organisation and rigour;**

  ➔ the Visual Assistant cannot help for this.

- **The ability to use finite windows on all sorts of large documents;**

  ➔ the Visual Assistant does not display a large document representing the stage through a small window but display the whole stage on the screen – no need to scroll the document.

- **To consider computers as a complement to – and not to exactly reproduce or substitute for – traditional artistic techniques offering hybrid ways of working;**

  ➔ the most artistic element of theatre lies not especially in the early stage – the concept formation – but in the most visual aspects of theatre as well as in the overall process of putting on a play. The Visual Assistant is just a sketching tool that takes place among a set of various tools and theatre practices.

Drawbacks…

- **A loss of sensitivity to space and light;**

  ➔ the lighting system in the Visual Assistant is very basic : one can make the whole scene lighter or darker whole scene or only make it only for the objects individually.

- **(repeated) A sense of good organisation and rigour;**

  ➔ the Visual Assistant cannot help for this as some creative practitioners feel hindered in their creative activities by too much organisation and rigour.

- **(repeated) The ability to use finite windows on all sorts of large documents;**

  ➔ the Visual Assistant does not display a large document representing the stage through a small window but display the whole stage on the screen – no need to scroll the document. The user does not feel confused by the only seeing a portion of a large document (a blueprint, for instance).

- **Metaphors can be perceived as obstacles to the observation of the software applications' marginal behaviour;**

  ➔ So far, we do not know what kind of marginal behaviours the artists/designers could discover and observe with the Visual Assistant. The main obstacle to fully appreciate the use of the Visual Assistant is in trying to get a photo-realistic quality in the visualisation/design of the plays.

Advantages…

- **Use of Internet to overcome space and time limits and collaborate on works of art or exhibitions;**

  ➔ the Visual Assistant at the moment has no functionalities for collaboration over open or private networks. This will most probably be considered for a future version of the Visual Assistant – it could possibly be the subject of another degree training and master's degree dissertation.

- **Great use of the UNDO command as it offers a means to reverse the effect of time;**

➔ at present, the Macintosh version has an UNDO function that can cancel the effects of the last action only. This function is unfortunately not available for all the functionalities.

- **Computers offer artists and designers possibilities to edit and print multiple versions of their works;**

➔ this is possible with any piece of software.

- **Interactivity is a very attractive feature of computers;**

➔ the objective of the Visual Assistant is not to produce a interactive work of art.

- **Use of the computer to reduce storing space and intelligently store documents in a digital form;**

➔ the Visual Assistant, in its Macintosh version, stores all the stages as a sequence that can be played like a slideshow. This solution can appear to the theatre practitioners as being more convenient than using a sheet of paper for each stage sets.

A tool for students in theatre production

The *Visual Assistant* has been used in several teaching workshops. This software is intended to be increasingly used for education purposes as it offers many advantages : ease-of-design, rapid prototyping and, last but not least, cheaper than setting theatre properties on an actual stage !

Students in theatre production have to learn how to design stage sets for a play. This is a rather costly activity, both in money and time. Indeed, to do it for real, each student should have at one's disposal a real theatre stage. To rent a theatre for a whole classroom, one student at a time, one stage set by one, is neither feasible nor affordable. It also means that the college should have real objects like furniture but also people or dummies for the students to figure out where to place everything and which space they need on the stage. It is time consuming to place all the objects, to change their place to improve the

arrangement of the properties – especially when this has to be done for a multi-stage play and for each student.

Therefore, an objective of the *Visual Assistant* is to save time and money by allowing the students to use a computer to virtually decorate a stage. By this action of 'sketching with the computer', they can get a preliminary idea of what they want. This is far less expensive than to do it for real ! They can design their stages and eventually do it life-sized if it is necessary.

Another goal the *Visual Assistant* is seeking after is to re-educate theatre students. In our present times, the ease of access to television and films is so great – even from early childhood – that many students conceive many aspects of the theatre in terms of television or films techniques. The 'close-up' desire is probably the best example of this observation. They also think about theatre mainly in terms of 'words' : scripts, reviews, ... Hence the need to develop their visual and spatial skills – which are actually the rudimentary skills of stage designers – and to make them realise "*that full scale theatre designs is often impractical and expensive*". The *Visual Assistant* offers such potentialities since it represents the stage as a whole, using 2D-images (rather like flat 'cardboard cut-outs') as theatre properties or actors and the possibility to explore spatiality by setting out objects in a 3D, virtual space. Moreover, that 'virtuality' appears as a cheap solution.

Overview of the functionalities[7]

The *Visual Assistant* (version 1.3 for the Macintosh) is an application that enables the user to collect textual and graphical material and to organise it visually. It builds upon the visual and spatial qualities of objects on the screen, rather than their formal (any semantic information such as object type, name, etc.), physical (location, size, image) or textual properties (any written comments). It enables the user to construct abstract or quasi-realistic representations – the latter ones could actually become real stage sets for real theatre plays.

Whilst there is 3D-modelling software to produce accurate 3D models of theatre layout and stage and set design (including such features as simulated lighting), such an approach

[7] [Beardon, 1999a]

assumes a knowledge of detail and a commitment of time that may not be available or even desirable when ideas are in formation. The *Visual Assistant* provides a quick way of creating the general impression generated by a stage, or other 3D space, so that the important concepts are clarified and can be discussed. "*The environment has a certain logic, but it is not based upon photo-realistic geometry. The uncompromising lack of photo-realism has been found by educators not to be a weakness but a strength of the software. It forces student users to express ideas approximately, yet in a way that creates what one practitioner has described as 'atmosphere'*" [Beardon, 1999b].

The fundamental architecture of the *Visual Assistant* is that it contains a sequence of 'stages', each of which contains a number of 'objects'.

Stages

Each stage is bounded by a three dimensional 'box' into which objects can be placed. Within the *Visual Assistant* this space may be viewed from the front or the top (as a saved VRML file it may be fully explored in 3D using a browser and the appropriate plug-in). All objects in the stage have a visual 2-dimensional representation, rather like a flat 'cardboard cut-out'. These objects can be arranged in the 3-dimensional space. One object can be put onto the floor of the space.

When viewed from above the various images appear as lines with width but minimal depth. They can be moved around in this top view and the moves are permanent, i.e. they take effect if the user changes back to front view. By moving the ends of objects in top view, objects can be rotated around their centre point (i.e. they need not face the front of the stage). Back to the front view, the moved objects appear greater or smaller if they have been moved forward or backward respectively. Objects rotated in the top view appear like a trapezoid in the front view – this to simulate a perspective effect due to a leaned object.

In the front view, one can see an outlined rectangle – representing the back of the stage – with four lines linking its corners with the corners of the display area (as opposed to the control area in which the 'Tool Box' is displayed). Together, they represent a 'box' bounding the stage. This 'box', called the 'Wireframe' can be switched on/off if a designed stage requires no spatial limits (e.g. when only the characters or some properties are lit by the spotlights and the rest lies in the dark).

30 stages exist as a sequence. An object created on any stage will exist on all subsequent stages unless it is changed or deleted. There is an option to 'play' the sequence of stages and this displays a simplified animation, rather like a slide show.

The commands available for stages are :

- go to a given stage,
- paste a colour to the floor,
- paste an object to the floor,
- change the image on the floor to a pattern,
- flip (horizontal or vertical) the image on the floor,
- re-colour the image on the floor,
- change the transparency of the object on the floor,
- remove the object from the floor,
- change the shape of the floor,
- make a scene lighter or darker,
- change the colour of a whole scene,
- change the background colour,
- play the stages,
- go to a top or front view.

Objects

Predefined images (from CD-ROMs, the Web, captured by digital cameras or through scanning, or through other applications) are entered as standard format files. They may also be entered by copying from other applications and pasting into the *Visual Assistant*. If the image is larger than the screen it will be reduced to fit the screen.

In addition, the user may create simple objects directly in the *Visual Assistant* - objects such as rectangles, circles, lines and pencil drawings.

Once entered the object can be manipulated by the following commands :

- import a new object,
- create a rectangle,
- create a circle,
- create a line,
- draw with a pencil,
- set pencil width,
- eraser (for use on any object),
- move an object (horizontal & vertical),
- move an object (horizontal & deep),
- resize an object,
- delete an object,
- duplicate an object,
- group two or more objects (i.e. make into a single object),
- change transparency of an object,
- flip an object (horizontal or vertical),
- rotate an object around its centre point (only in top view),
- make the image of an object into a pattern,
- resize the pattern,
- fix the pattern,

- re-colour an object,
- make an object lighter or darker,
- clean stray pixels,
- add or amend text associated with an object,
- paste object to floor.

Visual Assistant : the PC prototype

The *Visual Assistant* has been developed as part of the Leonardo da Vinci HaMLET[8] project. It has also received support from the NVRCAD[9] project funded under JISC/JTAP[10]. The PC prototype has been developed by myself as part of this master's degree dissertation.

For the specifications of the PC version of the Visual Assistant were in the form of the Macintosh version, I had not the opportunity to actually apply myself the principles of software design as exposed in the previous chapter. The observation of the user's behaviour in its context, the building of the ontology of the domain, going through a prototyping/testing cycle to write and review the specifications, all this work has been done by the author of the Macintosh version of the Visual Assistant, Colin Beardon. As a teacher, Colin had the chance to observe the students interacting with the technology, adopting a negative attitude towards the technology or, at the opposite, amazing themselves at the potential of computers. For instance, the early prototype versions of the Visual Assistant (in its Macintosh version) were used by educators, Theatre and Performance students from different Universities (Helsinki, Paris, Plymouth, Malmö,...) and these trials revealed, among others, that students were experiencing frustrations whenever they had to design real stage sets and were expecting photo-realistic quality. However, during all my degree training, my way of thinking (mostly influenced by the software engineering methods) was faced with the software design stake of Colin Beardon – my training advisor.

[8] 'Multimedia Learning Environment for Theatre and film'. Web site at http://vconf.hut.fi/hamlet/
[9] 'the Networked Virtual Reality Centres for Art & Design'. Web site at http://www.esad.plym.ac.uk/nvrcad/

This PC version does not meet all the specifications as given by the Macintosh version of the *Visual Assistant*. Only a few commands have been implemented so far. As an example of software supporting creative and working practices and a software for sketching ideas, this PC version, in my opinion, does not need to be fulfilled to illustrate our point in this master's degree dissertation. Further implementation of the *Visual Assistant* PC version could be undertaken as part of, possibly, another degree training in the Exeter School of Art and Design (ESAD).

Here is a list of the commands implemented in the PC prototype of the Visual Assistant I developed :

- look stage from a top view,
- look stage from a front view,
- import a new object,
- create a rectangle,
- create a circle,
- create a line,
- draw with a pencil,
- set pencil width,
- move an object (horizontal & vertical : toward the left/right or top/bottom sides of the 'box' delimiting the stage – only in front view),
- move an object (horizontal & deep : toward the left/right or back/front sides of the 'box' delimiting the stage – only in top view),
- delete an object,
- change transparency of an object.

[10] ' JISC (Joint Information Systems Committee) Technology Applications Programme'. Web site at http://www.jtap.ac.uk/

To make up for the commands not implemented in this version, one could use graphic packages – some of them are free of charge and can be downloaded from Internet or found on freeware CD-ROM – to change objects to greyscale or to colour them while retaining the light/dark shading of the original image; to make objects lighter or darker; to flip objects around a vertical or horizontal axis; to erase some parts of the objects or clean stray pixels, for instance. However, this does not go without two drawbacks :

- importing images in a graphic package and saving them before inserting them again in the *Visual Assistant* will appear as a tedious solution;
- although graphic packages offer more possibilities to retouch images, this with more details, they need a commitment of time to get a feel of such image-retouching tools.

A command such 'move deep' that should be available in the front view is not implemented though. A way to place an object at the back of the 'box' delimiting the stage is to go to the top view and move the object backwards from this standpoint.

It is unfortunately not possible to save the work done with the PC prototype. But this is not much a real problem in the sense that our purpose is not the one a student in theatre production could have. Presenting the *Visual Assistant* as an example for this master's degree dissertation, we do not really need to put our work aside and show it later to a teacher for getting marks. No `VA` or `VRML` file format has been implemented in the *Visual Assistant* prototype, hence it is not possible to open a previously saved *Visual Assistant* 'world' – i.e. a sequence of at least one stage set.

At the bottom of the screen, the *Visual Assistant* displays a control panel called 'Tool Box'. The ergonomy and the graphical aspects of that 'Tool Box' are not much to look at. Yet I used this as a temporary solution for developing the prototype and it certainly should be corrected towards the end of the development cycle. Because of this temporary situation, some buttons only display abbreviations or part of command words. A designer should be hired to drastically improve those aspects of the control panel. But that time has not come yet as the *Visual Assistant* still needs functional improvements, modifications in

the way a stage is being represented[11], and more testing from the students that are or will use it as part of their cursus.

User guide

Commands

In this section, the commands available in the prototype for the PC version of *Visual Assistant* are presented with a brief description. For each, the path through the menus and the name of the menu item are given with the following syntax :

[Menu\(Sub-Menu)\Menu Item]

Below this path you can find a brief description of the commands.

[Edit\Draw Shapes\Circle]

Creates a new circle. Click once with the left mouse button and hold it. Drag the shape until it reaches the desired size and release the button.

[Edit\Delete Object]

If you click on an object it will be permanently removed.

[Edit\Draw Shapes\Freehand Pencil]

Enables you to draw with a pencil. The thickness of the line can be changed by clicking on the Pen Size tool. The colour of the pencil is the current colour. Each time you release the Pencil tool you will create a new object. This new object can be moved as any imported object.

[Stage\Front View]

Changes the viewpoint to a position directly facing the centre of the back wall and at approximately two stage depths from the front of the stage.

[11] A theatre stage is actually deeper than larger – which is not the case in the current representation of a stage in both the Macintosh and PC versions of the *Visual Assistant*. Such a stage allows to place many backgrounds and theatre properties behind the one that is currently used for the performance. The foremost background just need to be raised for the public to see the one behind.

[Edit\Insert Object]

You will be asked to select a BMP file. A picture will appear centred on the screen (located at the front of the stage). The image of the object will be shown as "transparent" (i.e. all white pixels will be transparent).

[Edit\Draw Shapes\Line]

Creates a straight line in the current colour. The width of the line is determined by pen size.

[Edit\Move]

Enables you to move an object on a vertical plane from the front view. As the distance from the front of the stage is fixed, the object will not get larger or smaller.

If in Top View, then objects (represented by thin rectangles) can be moved around the stage by picking them up from inside the rectangle. When you return to Front View, the object will appear greater or smaller if it has been moved towards the front or the back of the stage respectively.

[File\New World]

Will create a new, empty environment. If the current environment has been altered since the last New, Load or Save command then users will be asked if they wish to save the present environment. It will then be deleted. Please note that saving the current world just creates an empty file (with the extension '.VA') since no VA or VRML file format has been implemented in the PC prototype.

[Edit\Pen Size]

Opens a dialogue window to reset the size of the pen (for Pencil and Line tools)

[Edit\Colour\Pick a Colour]

Shows the current colour and, if selected, opens the colour picker to select a new colour.

[Edit\Colour\Pick Black]

Sets current colour to black.

[Edit\Colour\Pick White]

Sets the current colour to white.

[File\Quit Visual Assistant]

Will quit the Visual Assistant and return to the desktop. If the current environment has been altered since the last New, Load or Save command then users will be asked if they wish to save the present environment before quitting. Please note that saving the current world just creates an empty file (with the extension '.VA') since no VA or VRML file format has been implemented in the PC prototype.

[Edit\Draw Shapes\Rectangle]

Creates a new rectangle. Click once with the left mouse button and hold it. Drag the shape until it reaches the desired size and release the button.

[Stage\Show Wireframe]

This item behaves like a toggle option : when the check mark is visible, a wireframe is displayed. Selecting the check-marked item turns off the mark and make the wireframe disappear.

[Stage\Top View]

This command permits viewing from a point directly on top of the stage. Objects will appear as thin grey rectangles outlined by a strip green line. When selected with the cursor, this rectangle representing the object in Top View may be dragged and dropped anywhere within the stage limits. The effect of changes made in 'Top View' will be permanent and will be reflected when you return to 'Front View'.

In this view, the command 'Move' is selected by default so that the user can save mouse displacements if she or he intended to move objects backwards or forwards on the stage.

[Edit\(Un)Make Transparent]

If you click on an object, the transparency of the image will change, i.e. if it is currently 'transparent' (all white pixels are invisible) it will become 'opaque' (i.e. all white pixels within a white rectangle will be shown) and conversely.

Guided tour

Now that the commands have been described, we will guide you through a quick design of a single stage set using the *Visual Assistant* prototype. For this, we need a few theatre properties, i.e. a few images that can be found free of charge – or at affordable prices – anywhere on the Web, on CD-ROM's or scanned by your own means. A few of them come with the *Visual Assistant* prototype. Select for instance a wall texture, curtains, a sky and an armchair, a table, and lastly a character or two.

Launch the *Visual Assistant* application and select the item 'Insert Object' in the 'Edit' menu. You can also click on the corresponding button (the one displaying 'Inser') in the 'Tool Box'. A dialog box will appear and you will be prompted to choose a directory and a file (only the standard BMP file format is recognised at the moment). For now, choose the wall texture. Click on the OK button or double-click on the chosen file to validate your choice. The image will appear centred on the screen, located at the front of the stage.

To place the wall image at the back of the stage, you need to go to 'Top View'. For this, select that item in the 'Stage' menu. Our image is now represented by a thin grey rectangle outlined by a strip green line and is placed at the front of the stage – which is actually the bottom of the display area. Click once on it and, while holding the left mouse-button, drag it to the back of the stage (the top of the screen). That change made in 'Top View' will be permanent and will be reflected when you return to 'Front View' by selecting this item in the same 'Stage' menu. Your wall is smaller as it is now located against the back of the stage, inside the outlined rectangle of the 'box'.

Now import the image of the sky and through the top view place it just before the wall. Load the window (command 'Insert Object' in the 'Edit' menu) and place it right in front

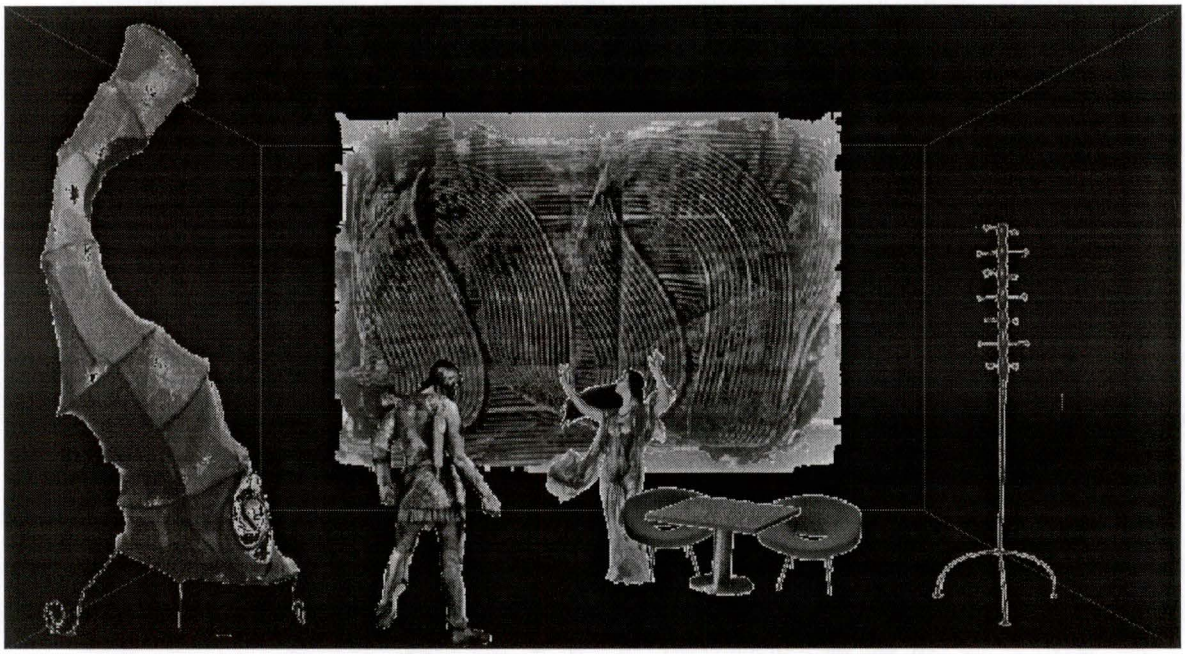
of the sky image. Load twice the image of the curtains, go to top view and place each of them besides the window. Back to the front view, adjust the position of the curtains : select the 'Move' item in the 'Edit' menu or on the 'Tool Box' and move the curtains vertically or horizontally.

The design of your stage is almost finished. You just need to insert the table, the chair and the armchair. In top view, place them wherever you want in the lowest third of the display area. And at long last, import the character(s) and place he/she/them among your properties.

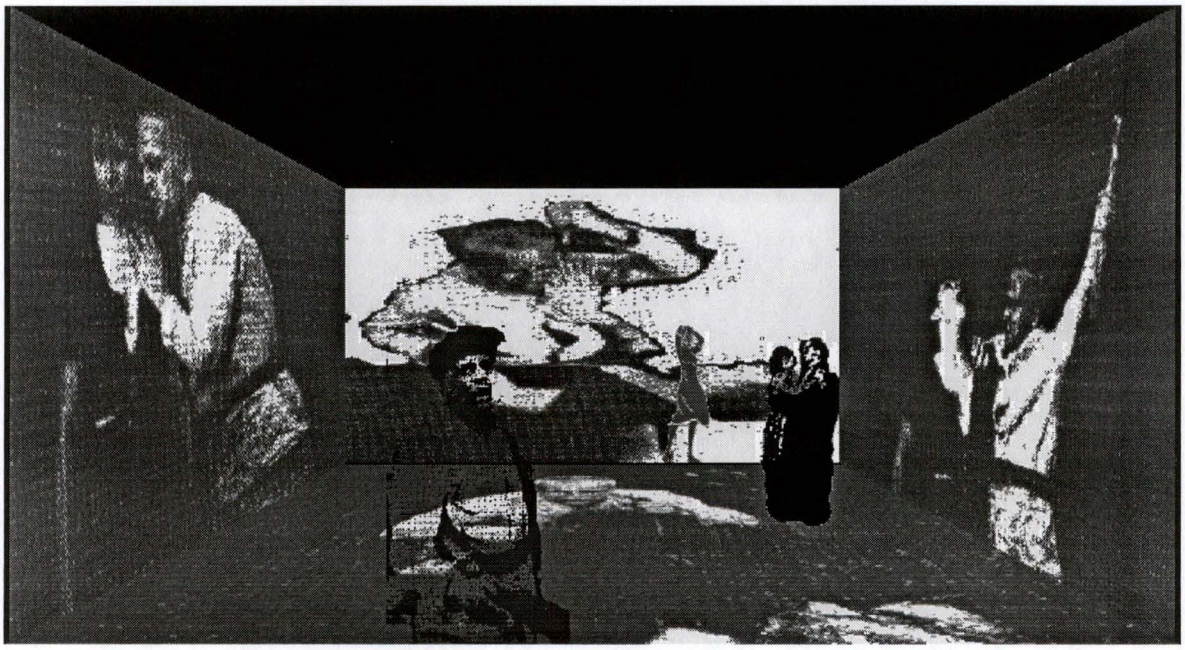
Note that if you need to resize an object, flip it vertically or horizontally or lean it to place it along the left/right sides of the stage, use a graphic package such as *Paint Shop Pro* (a shareware that can be found on Internet) or *PhotoShop* (commercial product) since those commands are not available in this version of *Visual Assistant* at the moment. We would like to apologise for imposing you such a tedious way of designing your stages. With an ounce of imagination, you can realise how easy it is to design a stage with a fully implemented version of the *Visual Assistant* compared with 3D packages (such as AutoDesk *3D Studio Max*) or 2D software (like Adobe *PhotoShop*).

Student works

The stage set designs shown below were made by students with the version of the Visual Assistant running on the Macintosh platform.




OFFICE



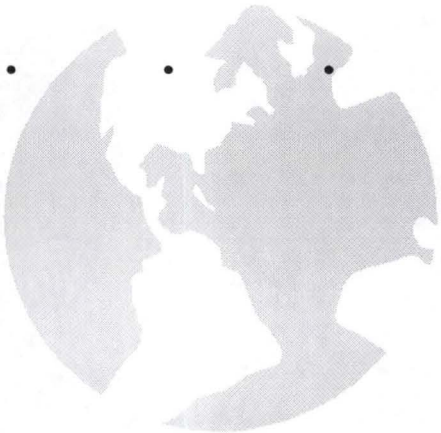
Conclusion

A sketching tool should pursue some design objectives (simple to learn and use, sketching rather than detailing, adequate for teaching and training, a good complement to traditional non computer-based techniques, etc.) to address the concerns evoked by artists and designers about software packages aimed at supporting their creative activities. Several of these design objectives are addressed by the Visual Assistant, which can be used by professionals as well as educators and students. The latter use is very important to demystify computers and prevent future creative professionals to develop a negative attitude towards the technology as the complementary use of computers can help in producing arresting and highly creative products, including plays – which is the concern of the Visual Assistant. We are convinced that the Visual Assistant is a well designed sketching tool for theatre practices and we hope that professionals and educators will make use of it – which would be the best proof of its adequacy to the practitioners and their domain of action.

MICHE Frédéric
Master's Degree of Computer Science,
5th Year

Master's degree dissertation

Conclusion



Main advisor : Jacques BERLEUR, s.j. (FUNDP) Training advisor : Colin BEARDON (ESAD)

Conclusion

Through this master's degree dissertation we first explained the unconscious thought (partly) understandable process of creativity. We took up a scientific approach to explain this phenomenon but do not think that this perspective dispelled the mystery of creativity.

We then observed the relationship between artists, designers and computer technology. We determined some of their needs and reproaches in a typology before reviewing the Visual Assistant in order to point out which demands have been fulfilled and which has not been.

We also hope to have proved the importance of computer-based sketching tools and of their contribution to the enhancement of the creative process in the domain of theatre but in any other creative field as well. This is important as computers will increasingly pervade more numerous everyday activities. Adopting a negative attitude towards the technology is a mistake and would amount to cut off oneself a bit more everyday or, less dramatic, would amount to unfairly criticise something not well understood. The use of the Visual Assistant and similar tools in education, according to us, should help to demystify computers and take apart this negative attitude. Through the prototype of the Visual Assistant I developed during my degree training, I hope to have contributed to extend that demystification and progressive disappearance of some negative attitude to the PC platform.

Computer-based sketching tools – such as the Visual Assistant – have proved their usefulness in the support of creative practices. But to improve the overall user experience with software technology is not sufficient. Therefore, we reviewed the software engineering methods and proposed an alternative to them which many computer professionals have named 'software design'. This discipline gives great consideration to the admittance of designers at an early stage of the process and to the necessity of listening to the practitioners of a domain as well as it means to effectively take into consideration the context of action in which the software application will take place to support creative practices. The application of such software development methods should lead to greater usability, reliability, dependability, appropriateness and better quality of the user's experience.

Software engineers must shift software development from a process of deriving a software system from specifications documents to a process of supporting the standard practices in a domain through software that enables more effective action. However, this shift in view will not occur overnight as many departments in universities are still teaching traditional software engineering techniques. Besides, students and professionals find far easier these latter as it requires some efforts to apply software design principles when you are not an expert in the field.

The originality of this master's degree dissertation lies not in revealing brand new software development methods – software design dates from the late eighties – but rather finds grounding in bringing together software design, creative practices in general, and theatre practices in particular. We hope that the Visual Assistant will serve as an incentive for software developers to develop similar applications in fields other than the one of theatre as well as to develop applications more human-centred in creative and any other domains.

We left the care to analyse the effects and consequences of collaborative design of stage sets over networks to further development of the Visual Assistant for PC platforms and of this master's degree dissertation. The Visual Assistant could be developed using a browser and Web page style as the interface. We also left the study of the impact of participatory design to further work. Participatory design techniques focus on enlisting practitioners of the relevant domain directly in the software design process – the domain of theatre with regard to this master's degree dissertation.

8MICHE Frédéric
Master's Degree of Computer Science,
5th Year

Master's degree dissertation

Bibliography

Main advisor : Jacques BERLEUR, s.j. (FUNDP) Training advisor : Colin BEARDON (ESAD)

Bibliography

[Beardon, 1999a]

Beardon Colin, *The Visual Assistant Web site*, last update on the 7th of May 1999. http://www.esad.plym.ac.uk/VA/index.html.

[Beardon, 1999b]

Beardon Colin, "The design of software to support creative practice", 1999. Paper to be presented at the *International Conference on Design and Technology Educational Research and Curriculum Development (IDATER 99)*, University of Loughborough, 23-25 August 1999. http://www.esad.plym.ac.uk/personal/C-Beardon/papers/paper9902.html.

[Beardon, 1999c]

Beardon Colin, Enright Terry, "The Visual Assistant: designing software to support creativity", 1999. Paper presented at *CADE'99 Conference*, University of Teesside, 5-7 April 1999 and to appear in the Proceedings. http://www.esad.plym.ac.uk/personal/C-Beardon/papers/paper9901.html.

[Beardon, 1998]

Beardon Colin, "Multimedia Learning Tools for the Theatre", 1998. Paper presented at the MESH Conference, London 1-2 October 1997. http://www.esad.plym.ac.uk/personal/C-Beardon/papers/9701.html.

[Beardon, 1997]

Beardon Colin, Gollifer Sue, Rose Christopher, Worden Suzette, "Computer Use by Artists & Designers", in M. Kyng and L. Mathiassen (Eds.) *Computers and Design in Context*, MIT Press, Cambridge (Mass.), 1997, pp. 27 – 50. This paper can be found at http://www.esad.plym.ac.uk/personal/C-Beardon/papers/9510.html under the title '*Designers as users*'.

[Boden, 1994]

Boden, Margaret A., "What is creativity ?", in Boden, Margaret A. et al., *Dimensions of creativity*, MIT Press, Cambridge (Mass.), 1994.

[Boden, 1992]

Boden, Margaret A., *The creative mind : myths and mechanisms*, Abacus, London, 1992 for the paperback (First published in 1990).

[Boehm, 1988]

Boehm Barry, "A spiral model of software development and enhancement", in *IEEE Computer* 21:2, May 1988.

[Crampton Smith-Tabor, 1996]

Crampton Smith Gillian, Tabor Philip, "The role of the artist-designer", in [Winograd et al., 1996].

[Denning-Dargan, 1996]

Denning Peter, Dargan Pamela, "Action-centred design", in [Winograd et al., 1996].

[Grand Robert, 1993]

Legrain Michel, Rey-Debove Josette, Rey Alain et al., *Le Grand Robert*, Dictionnaires Le Robert, Paris, 1993 (First published in 1967).

[Kapor, 1996]

Kapor Mitchell, "A software design manifesto", in [Winograd et al., 1996].

[Kelley-Hartfield, 1996]

Kelley David, Hartfield Bradley, "The designer's stance", in [Winograd et al., 1996].

[Liddle, 1996]

Liddle David, "Design of the conceptual model", in [Winograd et al., 1996].

[Longman, 1995]

Summers Della, Gadsby Adam, Rundell Michael et al., *Longman : dictionary of contemporary English*, Longman Group Ltd, Essex, 1995 (First published in 1978 then 1987).

[McCullough, 1998]

McCullough Malcolm, *Abstracting craft : The practiced digital hand*, MIT Press, Cambridge (Mass.), 1998 for the paperback (First published in 1996).

[Muller, 1993]

Muller Robert C., "Enhancing creativity, innovation and cooperation", in AI & Society (the journal of human-centred systems and machine intelligence), Volume 7 Number 1, Springer-Verlag, London, 1993.

[Quéau, 1993]

Quéau Philippe, *Le virtuel : vertus et vertiges*, collection milieux, éditions Champ Vallon/I.N.A., Seyssel, 1993

[Rheinfrank-Evenson, 1996]

Rheinfrank John, Evenson Shelley, "Design languages", in [Winograd et al., 1996].

[Somerville, 19??]

Somerville Ian, *Software engineering*, Addison Wesley, ?, 19??.

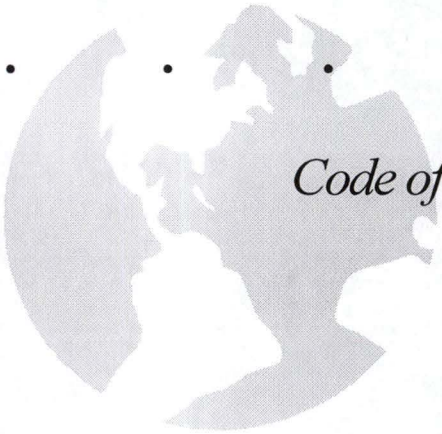
[Winograd et al., 1996]

Winograd Terry, Bennett John, De Young Laura, Hartfield Bradley et al., *Bringing design to software*, ACM Press, Baltimore (Md.), 1996.

MICHE Frédéric
Master's Degree of Computer Science,
5th Year

Master's degree dissertation

Appendix



Code of the Visual Assistant.

Main advisor : Jacques BERLEUR, s.j. (FUNDP) Training advisor : Colin BEARDON (ESAD)

Appendix

Code of the Visual Assistant.

Listing

We only listed after relevant parts of the code rather than every portion automatically generated by the 'C++Builder' Rapid Application Development (RAD) tool.

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année académique 1998-1999

Computer Sketching : How Software Tools Can Support Creative Practices ?

Annexes

Frédéric Miche

Mémoire présenté en vue de l'obtention du grade de « Maître en Informatique »

Computer Sketching : How Software Tools Can Support Creative Practices ?

Annexes

Frédéric Miche

Code of the PC Prototype of Visual Assistant

Vaprot.cpp (root file – execute code in *Main.cpp*)

```
//---------------------------------------------------------------------------

#include <vcl.h>

#pragma hdrstop

USERES("VAprot.res");

USEFORM("Main.cpp", MainForm);

USEFORM("Tools.cpp", ToolBox);

USEFORM("About.cpp", AboutBox);
USEFORM("Infos.cpp", InfosBox);
USEFORM("PenSizing.cpp", PenSizeForm);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    try
    {
        Application->Initialize();
        Application->Title = "Visual Assistant Prototype";
        Application->CreateForm(__classid(TMainForm), &MainForm);
        Application->CreateForm(__classid(TToolBox), &ToolBox);
        Application->CreateForm(__classid(TAboutBox), &AboutBox);
        Application->CreateForm(__classid(TInfosBox), &InfosBox);
        Application->CreateForm(__classid(TPenSizeForm), &PenSizeForm);
        Application->Run();
    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
    return 0;
}
//---------------------------------------------------------------------------
```

Main.h (Header file for *Main.cpp*)

```
//---------------------------------------------------------------------------
#ifndef MainH
#define MainH
//---------------------------------------------------------------------------
#include <vcl\sysutils.hpp>
#include <vcl\windows.hpp>
#include <vcl\messages.hpp>
#include <vcl\sysutils.hpp>
#include <vcl\classes.hpp>
#include <vcl\graphics.hpp>
#include <vcl\controls.hpp>
#include <vcl\forms.hpp>
#include <vcl\dialogs.hpp>
#include <vcl\stdctrls.hpp>
```

```
#include <vcl\buttons.hpp>
#include <vcl\extctrls.hpp>
#include <vcl\menus.hpp>
#include <Classes.hpp>
#include <Dialogs.hpp>
#include <Menus.hpp>
#include <Controls.hpp>
#include <ExtCtrls.hpp>
#include <ExtDlgs.hpp>
//---------------------------------------------------------------------------
enum TVATools {tsNoTool,
               tsNewWorld, tsOpenWorld, tsSave, tsSaveAs,
               tsTopView,
               tsInsert, tsRectangle, tsCircle, tsLine, tsFreehand, tsEraser,
tsPensize,
               tsMove, tsTransp,
               tsDelObj};
enum TWireframeMode {wfShow, wfHide};
enum TDisplayMode   {dmWhole, dmObjectClipRect_FrontView,
dmObjectClipRect_TopView, dmMergedClipRect};


class TMainForm : public TForm
{
__published:

      TMainMenu *MainMenu;
    TMenuItem *FileNewWorld;
    TMenuItem *FileOpenWorld;
    TMenuItem *FileSaveWorldAs;
    TMenuItem *FileQuit;
    TMenuItem *EditUndo;
    TMenuItem *EditCutObject;
    TMenuItem *EditCopyObject;
    TMenuItem *EditPasteObject;
    TMenuItem *HelpAbout;
    TMenuItem *Sep2;
    TMenuItem *FilePlay;
    TMenuItem *EditRedo;
    TMenuItem *Sep11;
    TMenuItem *EditMove;
    TMenuItem *EditMoveDeep;
    TMenuItem *Sep10;
    TMenuItem *EditDeleteObject;
    TMenuItem *StageMenu;
    TMenuItem *StageFrontView;
    TMenuItem *StageTopView;
    TMenuItem *StageGoToStore;
    TMenuItem *StageGoToWorkRoom;
    TMenuItem *Sep4;
    TMenuItem *StageCopyAll;
    TMenuItem *StageDelete;
    TMenuItem *Sep5;
    TMenuItem *StagePrior;
    TMenuItem *Sep3;
    TMenuItem *StageShowWireframe;
    TMenuItem *HelpVA;
    TMenuItem *StagePasteAll;
    TMenuItem *StageClear;
    TMenuItem *StageNew;
    TOpenDialog *OpenDialog;
    TSaveDialog *SaveDialog;
    TOpenPictureDialog *OpenPictureDialog;
    TMenuItem *StageLast;
```

```
TMenuItem *StageFirst;
TMenuItem *FileSaveWorld;
TMenuItem *StageObjectToWorkRoom;
TMenuItem *StageInfo;
TMenuItem *Sep9;
TMenuItem *EditEraser;
TMenuItem *EditResizeObject;
TMenuItem *EditFlipHorizontal;
TMenuItem *EditFlipVertical;
TMenuItem *EditPattern;
TMenuItem *EditResizePattern;
TMenuItem *EditTransparentObject;
TMenuItem *EditColour;
TMenuItem *EditColourPickColour;
TMenuItem *EditColourPasteColour;
TMenuItem *EditColourColourObject;
TMenuItem *EditColourPickWhite;
TMenuItem *EditColourPickBlack;
TMenuItem *EditObjectToFloor;
TMenuItem *WorkMenu;
TMenuItem *EditInsertObject;
TMenuItem *EditDrawShapes;
TMenuItem *EditShapeRectangle;
TMenuItem *EditShapeCircle;
TMenuItem *EditShapeLine;
TMenuItem *EditShapeFreeHandPencil;
TMenuItem *EditPenSize;
TMenuItem *WorkGoToStageFront;
TMenuItem *WorkGoToStageTop;
TMenuItem *Sep6;
TMenuItem *WorkBringToFront;
TMenuItem *WorkBringForward;
TMenuItem *WorkSendBackward;
TMenuItem *WorkSendToBack;
TMenuItem *Sep7;
TMenuItem *WorkClearWorkRoom;
TMenuItem *WorkTakeWorkToStage;
TMenuItem *StageNext;
TMenuItem *Sep1;
TMenuItem *Sep8;
TMenuItem *Sep12;
TColorDialog *ColorDialog;
TImage *ImageStage;
TBevel *Bevel1;
TMenuItem *EditCleanPixels;

void __fastcall FileQuitClick(TObject *Sender);
void __fastcall HelpAboutClick(TObject *Sender);
void __fastcall StageShowWireframeClick(TObject *Sender);
void __fastcall FileNewWorldClick(TObject *Sender);
void __fastcall FileOpenWorldClick(TObject *Sender);
void __fastcall FileSaveWorldAsClick(TObject *Sender);
void __fastcall EditUndoClick(TObject *Sender);
void __fastcall EditInsertObjectClick(TObject *Sender);
void __fastcall StageNewClick(TObject *Sender);
void __fastcall EditTransparentObjectClick(TObject *Sender);
void __fastcall EditColourPickWhiteClick(TObject *Sender);
void __fastcall EditColourPickBlackClick(TObject *Sender);
void __fastcall EditColourPickColourClick(TObject *Sender);
void __fastcall FileSaveWorldClick(TObject *Sender);
void __fastcall FilePlayClick(TObject *Sender);
void __fastcall StageFrontViewClick(TObject *Sender);
void __fastcall StageTopViewClick(TObject *Sender);
void __fastcall StageGoToStoreClick(TObject *Sender);
```

```
    void __fastcall StageGoToWorkRoomClick(TObject *Sender);
    void __fastcall StageObjectToWorkRoomClick(TObject *Sender);
    void __fastcall StageClearClick(TObject *Sender);
    void __fastcall StageCopyAllClick(TObject *Sender);
    void __fastcall StagePasteAllClick(TObject *Sender);
    void __fastcall StageDeleteClick(TObject *Sender);
    void __fastcall StageInfoClick(TObject *Sender);
    void __fastcall StagePriorClick(TObject *Sender);
    void __fastcall StageNextClick(TObject *Sender);
    void __fastcall StageFirstClick(TObject *Sender);
    void __fastcall StageLastClick(TObject *Sender);
    void __fastcall WorkGoToStageFrontClick(TObject *Sender);
    void __fastcall WorkGoToStageTopClick(TObject *Sender);
    void __fastcall WorkBringToFrontClick(TObject *Sender);
    void __fastcall WorkBringForwardClick(TObject *Sender);
    void __fastcall WorkSendBackwardClick(TObject *Sender);
    void __fastcall WorkSendToBackClick(TObject *Sender);
    void __fastcall WorkClearWorkRoomClick(TObject *Sender);
    void __fastcall WorkTakeWorkToStageClick(TObject *Sender);
    void __fastcall EditRedoClick(TObject *Sender);
    void __fastcall EditShapeRectangleClick(TObject *Sender);
    void __fastcall EditShapeCircleClick(TObject *Sender);
    void __fastcall EditShapeLineClick(TObject *Sender);
    void __fastcall EditShapeFreeHandPencilClick(TObject *Sender);
    void __fastcall EditPenSizeClick(TObject *Sender);
    void __fastcall EditEraserClick(TObject *Sender);
    void __fastcall EditCleanPixelsClick(TObject *Sender);
    void __fastcall EditMoveClick(TObject *Sender);
    void __fastcall EditMoveDeepClick(TObject *Sender);
    void __fastcall EditResizeObjectClick(TObject *Sender);
    void __fastcall EditFlipHorizontalClick(TObject *Sender);
    void __fastcall EditFlipVerticalClick(TObject *Sender);
    void __fastcall EditPatternClick(TObject *Sender);
    void __fastcall EditResizePatternClick(TObject *Sender);
    void __fastcall EditColourPasteColourClick(TObject *Sender);
    void __fastcall EditColourColourObjectClick(TObject *Sender);
    void __fastcall EditCutObjectClick(TObject *Sender);
    void __fastcall EditCopyObjectClick(TObject *Sender);
    void __fastcall EditPasteObjectClick(TObject *Sender);
    void __fastcall EditDeleteObjectClick(TObject *Sender);
    void __fastcall EditObjectToFloorClick(TObject *Sender);
    void __fastcall HelpVAClick(TObject *Sender);

/***** EVENT Functions ************************************************/
    void __fastcall ImageStageMouseDown(TObject *Sender,
          TMouseButton Button, TShiftState Shift, int X, int Y);
    void __fastcall ImageStageMouseMove(TObject *Sender, TShiftState Shift,
          int X, int Y);
    void __fastcall ImageStageMouseUp(TObject *Sender, TMouseButton Button,
          TShiftState Shift, int X, int Y);
    void __fastcall FormCreate(TObject *Sender);



private:        // private user declarations

/***** Functions for INITIALIZING *************************************/
    void __fastcall CreateWorld();  //A "World" is a list of Stages
    void __fastcall CreateStage();  //A "Stage" is a list of Objects
    void __fastcall DeleteWorld();
    void __fastcall DeleteStage();
/***** MISC. : Functions for SAVING, UPDATING, etc. ********************/
    TModalResult __fastcall CheckSave();
    void __fastcall UpdateFunctionsAvailable();
```

```
    void __fastcall WrappingUp();
    void __fastcall SortStageObjects();
/***** Functions for SELECTING *********************************************/
    void __fastcall SelectObject_FRONT(int X, int Y);
    void __fastcall SelectObject_TOP(int X, int Y);
/***** Functions for REFRESHING the FRONT VIEW ******************************/
    void __fastcall Rebuild_FRONT(TDisplayMode DisplayMode);
    void __fastcall CreateBlankStage_OffScr();
    void __fastcall S_H_Wireframe_OffScr();
    void __fastcall PaintObjects_OffScr(TDisplayMode DisplayMode);
    void __fastcall Display_OnScr(TDisplayMode DisplayMode);
/***** Functions for REFRESHING the TOP VIEW ********************************/
    void __fastcall Rebuild_TOP(TDisplayMode DisplayMode);
    void __fastcall CreateBlankFloor_OffScr();
    void __fastcall DrawTopRects_OffScr(TDisplayMode DisplayMode);
    //void __fastcall Display_OnScr(TDisplayMode DisplayMode);  --> The same
method is used to display what's off-screen
/***** Functions for DRAWING SHAPES *****************************************/
    void __fastcall DrawTempShape(POINT MouseLocOnClick, POINT MouseLocOnRelease);
    void __fastcall StoreShape   (POINT MouseLocOnClick, POINT MouseLocOnRelease);
/***** Functions for MOVING *************************************************/
    void __fastcall MergeRects_FRONT();
    void __fastcall MergeRects_TOP();
    bool __fastcall IntersectRects(TRect ComparedRect, TRect ModelOfComparison);
   TRect __fastcall GetObjectNewCoord_FRONT(int X, int Y, int XOffset, int
YOffset);
   TRect __fastcall GetObjectNewCoord_TOP(int X, int Y, int XOffset, int YOffset);
/***** Functions for VA's 3D-SPACE ******************************************/
   TRect __fastcall DepthRendering();
     int __fastcall ConvertLocation_3Dto2D(int MiddleLine, int HorizCoord, int
VertCoord);
     int __fastcall ConvertLocation_2Dto3D(int MiddleLine, int HorizCoord, int
VertCoord);
    void __fastcall CheckBounds(TRect ValidLocation_3D);


public:         // public user declarations
     virtual __fastcall TMainForm(TComponent* Owner);

/***** GENERAL information for WORLD, STAGES & OBJECTS
*****************************/
    bool                WorldModified;                  //Has the world been
modified since the last saving ?
    TModalResult        OpCancelled;                    //Has the user cancelled
the last [QUIT / NEW / OPEN] operation ?
    TList*              ListOfObjects;                  //A stage is made up of
objects stored in a list.
    int                 LastObjectID;                   //Global variable to
generate ID numbers for StageObjects
    int                 ObjectToSelect;                 //Used to extract form
ListOfObjects the object on which the current tool will be applied
    TVATools            PrevTool, CurTool;              //Data members to hold
current and previous tools
    int                 UserToScreenDist;               //Measures the distance
between the user and the screen's surface
/***** FRONT VIEW information for OFF-SCREEN & STAGE-SET
*************************/
    Graphics::TBitmap*  Workbench;                      //Off-screen bitmap
    TColor              StageColor, FloorColor;         //Keeps trace of the color
currently used for the Stage and the floor
    TWireframeMode      WireframeMode;                  //Tells if the Wireframe
has to be shown or hidden
/***** FRONT VIEW information for DRAWING
*****************************************/
```

```
    bool                CanDraw, CanDrawFreehand;       //Can I draw ? Can I do it
freehanded ?
    POINT               Origin, MovePt;                 //Structure with X and Y
fields to draw by dragging the mouse
    TRect               FreeHClipRect;                  //Clipping rectangle used
to draw with pencil (freehand)
    int                 CurrentPenSize;                 //Gives the size to use
when setting the Pen's width
/***** FRONT VIEW information for MOVING
*****************************************/
    bool                CanMove;                        //Can I move the object
I've selected ?
    TRect               MergedRectangle;                //Rect that includes the 2
rects defined here above
    int                 XOffset, YOffset;               //Distance between the
MouseLoc.[X/Y] and the left/top side of the clip.rect.,
                                                        //  but in TOP VIEW,
YOffset is the distance between the MouseLoc.[Y] and the object's [Center_3D.z] -
and not the top side anymore
};
//---------------------------------------------------------------------------

class TStageObject
{
public:           // User declarations
    __fastcall TStageObject();        //Constructor

/***** USER TYPES ****************************************/
typedef struct {
    int x, y, z;
}Point_3D;
/***** GENERAL information *******************************/
    int                 ID;                             //Identification number
(of course)
    Graphics::TBitmap*  StorageBmp;                     //Bitmap storing an object
of the stage
    //To get the object's transparency,        read the [TBitmap->Transparent]
property
    //To get the object's ORIGINAL dimensions, read the [TBitmap->Width] and [ -
>Height] properties
/***** VA'S 3D SPACE information *************************/
    Point_3D            Center_3D;                      //Coordinates of the
center in 3D space
/***** FRONT VIEW information ****************************/
    TRect               PrevPos_FRONT, NewPos_FRONT;    //Position ON THE SCREEN
area representing the stage (and not in VA's 3D-space) of the clipping rectangles
needed to move (deep) the object
    int                 CurWidth, CurHeight;            //Object's CURRENT
dimensions, that is the (shrinked) dimensions of the object's projection on the
screen's surface
/***** TOP VIEW information ******************************/
    TRect               PrevPos_TOP, NewPos_TOP;        //Position ON THE SCREEN
area representing the stage (and not in VA's 3D-space) of the clipping rectangles
needed to move (deep) the object

};
//---------------------------------------------------------------------------
extern PACKAGE TMainForm *MainForm;
//---------------------------------------------------------------------------
#endif
```

Main.cpp (master file calling all the other modules)

```
//---------------------------------------------------------------------------

#include <vcl\vcl.h>

#include <stdlib.h>


#pragma hdrstop


#include "Main.h"

#include "Tools.h"

#include "About.h"
#include "Infos.h"
#include "PenSizing.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"

TMainForm *MainForm;
/*****************************************************************************
**************************/
/* Instantiates the class used to store one of the objects on stage.
*/
/* New memory will be allocated whenever it's necessary.
*/
/* Can't make it work if it's defined elsewhere like this : [TStageObject*
StageObject = new TStageObject;] */
/*      in the code, where its' needed.  So it's defined here - as sort of a
global variable.                */
/*****************************************************************************
**************************/
TStageObject* StageObject;

/***** LEGEND
*****************************************************************************
*************/
/*              Symbol   |  Meaning
*/
/*-----------------------------------------------------------------------------
--------------------------*/
/*                  //   |  Line comment, permanent : do not remove.
*/
/*             //*****   |  Comment for guidance, to remove later on.
*/
/*            [] or ''   |  It's used to make it easier to spot data members' or
methods' names, or values   */
/*****************************************************************************
**************************/

/***** USAGE OF [Center_3D.z]
*****************************************************************************/
/*
*/
/* The point of origin for the axes in the 3D-SPACE is the Top-Left-Front corner.
When an object's         */
/*      [Center_3D.z] = 0, the object is at the front of the stage (which is a 3D-
space).  But in           */
```

```
/*      TOP VIEW, the Z-coordinate is starting at 0/the top of the screen area
representing the stage       */
/*      - which is the back of the stage in the 3D-space.
*/
/* That's why the value stored in [Center_3D.z] is '0' when the object is at the
front of the stage,        */
/*      that is the bottom of the screen area delimiting the stage (coord.
'ImageStage->Height -1').        */
/*      The value stored in [Center_3D.z] is 'ImageStage->Height -1' when the
object is at the back         */
/*      of the stage, that is the top of the screen area delimiting the stage
(coord. '0').                 */
/* To use [Center_3D.z] for setting coord on screen, we have to take [ImageStage-
>Height - 1] from          */
/*      which we substract [Center_3D.z] to get the displacement from the top of
the screen.                   */
/*
*/
/* N.B: Whatever the view is, no matter if there's a (perspective) depth effect,
the center we are          */
/*      considering is always the center in VA's 3D-space - and not the center of
the projection            */
/*      of the object in one of the views !
*/
/*****************************************************************************
**************************/

//---------------------------------------------------------------------------
__fastcall TMainForm::TMainForm(TComponent* Owner)
      : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::FormCreate(TObject *Sender)
{
//PART I : Creates the World
    //Calls the method that creates a new list of Stages - that is, a new World
    CreateWorld();

//PART II : Displays the World
    //WireframeMode is initialized here to show the wireframe when VA starts
    WireframeMode = wfShow;
    //Calls the method / member function that draws a blank stage + a wireframe
    Rebuild_FRONT(dmWhole);
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::CreateWorld()
/*****************************************************************************/
/* A "World" is a list of Stages                                             */
/* In this method, everything concerning one world is initialized here.      */
/*      That world is made up of only one stage and will be expanded later   */
/*****************************************************************************/
{
    //Sets the size of ImageStage
    ImageStage->Width  =  MainForm->ClientWidth;
    ImageStage->Height = (MainForm->ClientWidth * 0.51);  //Height is 51% of the
Width

    //Declares the bitmap representing the stage off-screen
    Workbench   = new Graphics::TBitmap;

    //Gives Workbench (offscreen bitmap) the size of a stage
```

```
    Workbench->Width  = ImageStage->Width;
    Workbench->Height = ImageStage->Height;

    //Sets a few other data members - e.g. CanMove, CurTool, ...
    WorldModified = false;                          //Sets Modified to false to
check later if work needs to be saved when user chooses NEW or OPEN
    LastObjectID = 0;                               //No objects at all on the
stage at this time
    ObjectToSelect = -1;                            //No object in ListOfObjects
has to undergo any operation at this time
    PrevTool = CurTool = tsNoTool;                  //No tool is selected at the
time of creation
    UserToScreenDist = 600;                         //The distance between the user
and the screen's surface is 600 units
    ColorDialog->Color = clTeal;                    //We need a default color for
drawing shapes
    StageColor = FloorColor = clNavy;               //clNavy is the default color
used for the Stage and the Floor
    CurrentPenSize = 3;                             //Default Pen Size for drawing
    CanDraw = CanDrawFreehand = CanMove = false;    //Sets all boolean to false -
safety measure
    Origin = MovePt = Point(0,0);                   //Starting coordinates
    XOffset = YOffset = 0;                          //No offset at the time of
creation

    //Calls the method that creates a new list of StageObjects - that is, a new
Stage
    CreateStage();

    //Updates the list of stages in Stage menu.

}
//---------------------------------------------------------------------------

void __fastcall TMainForm::CreateStage()
/*****************************************************************************/
/* A "Stage" is a list of Objects.                                           */
/* In this method, everything concerning one and only one stage that is,     */
/*      a list of objects, is initialized here.                              */
/*****************************************************************************/
{
    //A stage is made up of objects stored in a list.  At the time of creation,
only one empty list of objects is needed for the new world.
    ListOfObjects = new TList;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::DeleteWorld()
{
    //The off-screen bitmap has(?) to be deleted 'cos mem. has been dynamically
allocated to it - it doesn't exist at the design time.
    delete Workbench;

//***** Should loop through all the stages and delete them one by one by calling
//*****     [DeleteStage] with a argument indicating which Stage has to be deleted
    DeleteStage();
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::DeleteStage()
{
    //Frees the memory used by the objects listed in the TList of one stage.
    for (int i = 0; i < ListOfObjects->Count; i++)
    {
```

```
        StageObject = (TStageObject*) ListOfObjects->Items[i];
        delete StageObject;
    }
    //Then deletes the list itself.
    delete ListOfObjects;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::FileQuitClick(TObject *Sender)
{
    //Saves the current world if it's necessary.
    OpCancelled = CheckSave();

    if (OpCancelled != mrCancel)
    {
        //Calls the method that destroys all the Stages - that is, the World
        DeleteWorld();

        Application->Terminate();

    }//ELSE : do not quit VA because the user has cancelled the operation
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::FileNewWorldClick(TObject *Sender)
{
//***** Modify the LastStageObjectID

    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + FileNewWorld->Hint;
    PrevTool = CurTool;
    CurTool  = tsNewWorld;

    //Saves the current world if it's necessary.
    OpCancelled = CheckSave();

    if(OpCancelled != mrCancel)
    {
        //Disposes of the existing world.
        DeleteWorld();

        //Inits a new world made up of a blank stage only
        CreateWorld();
        //We need a default color for drawing shapes
        ToolBox->ToolBtnPickColour->Color = clTeal;     //It can't be done in
[createWorld] 'cos the ToolBox is not created first at the time of creation -
which results in an "ACCESS VIOLATION"

        //We don't need to save a blank world --> wait for further changes
        WorldModified = false;

        //Clears the screen and shows the Front View
        Rebuild_FRONT(dmWhole);
        ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text = " Stage - Front View";

        //Tells the user that a new world has been created and displayed
        ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " New World created
and displayed";

        //A few functionalities have to be (dis)activated in Front View
        UpdateFunctionsAvailable();

        //Empties the list of stages in Stage menu.
```

```
        //It's a new world, so we don't know its name yet
        SaveDialog->FileName = "";

    }//ELSE: do not clear the current World because the user has cancelled the
operation
    else
        //Tells the user that 'Create New World' has been cancelled
        ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " 'Create New World'
has been cancelled";
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::FileOpenWorldClick(TObject *Sender)
{
//***** Modify the LastStageObjectID

    //Modifies ToolBox appearance to reflect click on function  and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + FileOpenWorld->Hint;
    PrevTool = CurTool;
    CurTool  = tsOpenWorld;

    //Saves the current world if it's necessary.
    OpCancelled = CheckSave();

    if(OpCancelled != mrCancel)
    {
        //Disposes of the existing world.
        DeleteWorld();

        //Inits a new world made up of a blank stage only
        CreateWorld();
        //We need a default color for drawing shapes
        ToolBox->ToolBtnPickColour->Color = clTeal;     //It can't be done in
[createWorld] 'cos the ToolBox is not created first at the time of creation -
which results in an "ACCESS VIOLATION"

        //Expands the blank world created here above with information loaded from
a VA file
        OpenDialog->FileName = "";
        if (OpenDialog->Execute())
        {
            SaveDialog->FileName = OpenDialog->FileName;
            if (MessageDlg("Under construction - No loading at the moment ! \nOnly
a blank World is created.", mtWarning,
                    TMsgDlgButtons() << mbOK, 0) == mrOk)
                    //does nothing
                    ;
        }

        //We don't need to save a world thas has been loaded and, a fortiori,
previously saved --> wait for further changes
        WorldModified = false;

        //Clears the screen and displays in FRONT VIEW the World that has just
been loaded
        Rebuild_FRONT(dmWhole);
        ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text = " Stage - Front View";

        //Tells the user that a world has been loaded and displayed
        ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " A World has been
loaded and displayed";
```

```
        //A few functionalities have to be (dis)activated in Front View
        UpdateFunctionsAvailable();

        //Renews the list of stages in Stage menu.

    }//ELSE : do not clear the current World nor load a new World because the user
has cancelled the operation
    else
        //Tells the user that 'Open World' has been cancelled
        ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " 'Open World' has
been cancelled";
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::FileSaveWorldClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + FileSaveWorld->Hint;
    PrevTool = CurTool;
    CurTool  = tsSave;

    if (SaveDialog->FileName == "")
                FileSaveWorldAsClick(Sender);
    else
    {
        //SaveToFile();

        //Sets Modifed to false since we've just saved
        WorldModified = false;

        if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
              //does nothing
              ;

        //Tells the user that the world has been saved
        ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " The World has been
saved";
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::FileSaveWorldAsClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + FileSaveWorldAs-
>Hint;
    PrevTool = CurTool;
    CurTool  = tsSaveAs;

    if (SaveDialog->Execute())
    {
        //SaveToFile();

        //Sets Modifed to false since we've just saved
        WorldModified = false;

        if (MessageDlg("Under construction. \nJust believe that your world has
been saved !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
        //does nothing
```

```
        ;

        //Tells the user that the world has been saved with a new name
        ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " The World has been
saved as : '" + SaveDialog->FileName + "'";
    }
    else
        //Tells the user that the saving has been cancelled
        ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " Saving has been
cancelled";
}
//---------------------------------------------------------------------------

TModalResult __fastcall TMainForm::CheckSave()
{
    if (WorldModified)
    {
        switch(MessageDlg ("The current world has been changed. \nSave changes ?",
                           mtConfirmation, TMsgDlgButtons() <<mbYes <<mbNo
<<mbCancel, 0))
        {
            case mrYes      :
                //If YES was clicked, then save the world;
                FileSaveWorldAsClick(this);
                return mrYes;
            case mrNo       :
                //If NO was clicked, then don't save - just do nothing;
                return mrNo;
            case mrCancel   :
                //Tells the user that the saving has been cancelled
                ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " Saving has
been cancelled";

                //If CANCEL was clicked, then return - the return value is used in
[QUIT] to be able to cancel it.
                return mrCancel;
        }
    }
    //By default (and to avoid 'Compiler Warning'), the function returns [mrNone]
    return mrNone;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditUndoClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditUndo->Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditInsertObjectClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditInsertObject-
>Hint;
    PrevTool = CurTool;
    CurTool  = tsInsert;
```

```
    if (OpenPictureDialog->Execute())
    {
        //Loads the image on a temporary container (Type of it : TImage, in order
to use Autosize)
        TImage* TempImage = new TImage(this);
        TempImage->AutoSize = true;
        TempImage->Picture->LoadFromFile(OpenPictureDialog->FileName);

        //Allocates a new memory space for that object on stage and returns a
pointer to it
        StageObject = new TStageObject;
        //Gives an ID number to the object
        StageObject->ID = ++LastObjectID;

        //The bitmap's height and width properties hold the original size of the
object
        StageObject->StorageBmp->Width  = TempImage->Picture->Width;
        StageObject->StorageBmp->Height = TempImage->Picture->Height;

        //When an object is created, its current dimensions are identical to its
original dimensions
        StageObject->CurWidth  = StageObject->StorageBmp->Width;
        StageObject->CurHeight = StageObject->StorageBmp->Height;

        //Places and stores the position of the image on the stage
        StageObject->NewPos_FRONT.Left   = (ImageStage->Width  / 2) - (TempImage-
>Picture->Width  / 2);
        StageObject->NewPos_FRONT.Top    = (ImageStage->Height / 2) - (TempImage-
>Picture->Height / 2);
        StageObject->NewPos_FRONT.Right  = (ImageStage->Width  / 2) + (TempImage-
>Picture->Width  / 2);
        StageObject->NewPos_FRONT.Bottom = (ImageStage->Height / 2) + (TempImage-
>Picture->Height / 2);

        //Records the center in VA's 3D-space
        StageObject->Center_3D.x = ImageStage->Width  / 2;
        StageObject->Center_3D.y = ImageStage->Height / 2;
        StageObject->Center_3D.z = 0;

        //Copies Image from temp container to StorageBmp
        StageObject->StorageBmp->Assign(TempImage->Picture->Bitmap);
        delete TempImage;

        //Adds StageObject to the list of objects present on stage
        ListOfObjects->Add(StageObject);

        //Sets [ObjectToSelect] to make it reference to the image that has just
been inserted
        ObjectToSelect = (ListOfObjects->Count) - 1;       //It's the last index
because the list hasn't been sorted since

        //Display the image that has just been loaded off- and on screen
        Rebuild_FRONT(dmObjectClipRect_FrontView);

        //As a safety measure...
        WrappingUp();
    };
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageShowWireframeClick(TObject *Sender)
{
```

```
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageShowWireframe-
>Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    // Shows / Hides the wireframe
    if (StageShowWireframe->Checked)
    {
        //Unchecks the Wireframe item in the menu
        StageShowWireframe->Checked = false;

        WireframeMode = wfHide;
    }
    else
    {
        //Checks the Wireframe item in the menu
        StageShowWireframe->Checked = true;

        WireframeMode = wfShow;
    };
    Rebuild_FRONT(dmWhole);
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageNewClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageNew->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    //Adds a new stage to the world.

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
            TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditTransparentObjectClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBtnTransparentObject->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " +
EditTransparentObject->Hint;
    PrevTool = CurTool;
    CurTool  = tsTransp;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditColourPickWhiteClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditColourPickWhite-
>Hint;
    ToolBox->ToolBtnPickColour->Color = clWhite;
```

```
    ColorDialog->Color = clWhite;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditColourPickBlackClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditColourPickBlack-
>Hint;
    ToolBox->ToolBtnPickColour->Color = clBlack;
    ColorDialog->Color = clBlack;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditColourPickColourClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " +
EditColourPickColour->Hint;

    if (ColorDialog->Execute())
    {
        ToolBox->ToolBtnPickColour->Color = ColorDialog->Color;
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::FilePlayClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + FilePlay->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageFrontViewClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageFrontView-
>Hint;
    ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text = " Stage - Front View";
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    //Shows the Front View
    Rebuild_FRONT(dmWhole);

    //A few functionalities have to be (dis)activated in Front View
    UpdateFunctionsAvailable();

    //No function selected by default even though is the other way round in TOP
VIEW.
    ToolBox->ToolBtnMove->Down = false;
```

```
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " Please, choose a
function";
    PrevTool = CurTool;
    CurTool = tsNoTool;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageTopViewClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageTopView->Hint;
    ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text = " Stage - Top View";
    PrevTool = CurTool;
    CurTool  = tsTopView;

    //Shows the Top View
    Rebuild_TOP(dmWhole);

    //A few functionalities have to be (dis)activated in Top View
    UpdateFunctionsAvailable();

    //The function selected by default is [Move].
    ToolBox->ToolBtnMove->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditMove->Hint;
    PrevTool = CurTool;
    CurTool = tsMove;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageGoToStoreClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageGoToStore-
>Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageGoToWorkRoomClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageGoToWorkRoom-
>Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageObjectToWorkRoomClick(TObject *Sender)
```

```
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " +
StageObjectToWorkRoom->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageClearClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageClear->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageCopyAllClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageCopyAll->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StagePasteAllClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StagePasteAll->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageDeleteClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
```

```
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageDelete->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageInfoClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageInfo->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StagePriorClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StagePrior->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageNextClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageNext->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageFirstClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageFirst->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;
```

```
    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageLastClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageLast->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::WorkGoToStageFrontClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageFrontView-
>Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    //Carry out a few things if necessary then call StageFrontView

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::WorkGoToStageTopClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageTopView->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    //Carry out a few things if necessary then call StageTopView

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::WorkBringToFrontClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnBringToFront->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + WorkBringToFront-
>Hint;
```

```
    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::WorkBringForwardClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnBringForward->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + WorkBringForward-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::WorkSendBackwardClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnSendBackward->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + WorkSendBackward-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::WorkSendToBackClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnSendToBack->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + WorkSendToBack-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::WorkClearWorkRoomClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + WorkClearWorkRoom-
>Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------
```

```
void __fastcall TMainForm::WorkTakeWorkToStageClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + WorkTakeWorkToStage-
>Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditRedoClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditRedo->Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditShapeRectangleClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnRectangle->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditShapeRectangle-
>Hint;
    PrevTool = CurTool;
    CurTool = tsRectangle;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditShapeCircleClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnCircle->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditShapeCircle-
>Hint;
    PrevTool = CurTool;
    CurTool = tsCircle;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditShapeLineClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnLine->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditShapeLine->Hint;
    PrevTool = CurTool;
    CurTool = tsLine;
```

```
    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditShapeFreeHandPencilClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnFreehandPencil->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " +
EditShapeFreeHandPencil->Hint;
    PrevTool = CurTool;
    CurTool = tsFreehand;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditPenSizeClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditPenSize->Hint;
    PrevTool = CurTool;
    CurTool = tsPensize;

    if (PenSizeForm->ShowModal() == mrOk)
    {
        CurrentPenSize = StrToInt(PenSizeForm->EditSize->Text);
    }
    else
    {   //keep the same Pen Size
        PenSizeForm->EditSize->Text = CurrentPenSize;
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditEraserClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnEraser->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditEraser->Hint;
    PrevTool = CurTool;
    CurTool = tsEraser;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditCleanPixelsClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnCleanPixels->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditCleanPixels-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
```

```
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditMoveClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnMove->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditMove->Hint;
    PrevTool = CurTool;
    CurTool = tsMove;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditMoveDeepClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnMoveDeep->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditMoveDeep->Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditResizeObjectClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnResizeObject->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditResizeObject-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditFlipHorizontalClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnFlipHorizontal->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditFlipHorizontal-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditFlipVerticalClick(TObject *Sender)
{
```

```
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnFlipVertical->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditFlipVertical-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditPatternClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnPattern->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditPattern->Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditResizePatternClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnResizePattern->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditResizePattern-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditColourPasteColourClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnPasteColour->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " +
EditColourPasteColour->Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditColourColourObjectClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
```

```
    ToolBox->ToolBtnColourObject->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " +
EditColourColourObject->Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditCutObjectClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnCutObject->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditCutObject->Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditCopyObjectClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnCopyObject->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditCopyObject-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditPasteObjectClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditPasteObject-
>Hint;
    ToolBox->ToolBtnPasteObject->Down = true;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditDeleteObjectClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBtnDeleteObject->Down = true;
```

```
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditDeleteObject-
>Hint;
    PrevTool = CurTool;
    CurTool  = tsDelObj;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditObjectToFloorClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnObjectToFloor->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditObjectToFloor-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------


void __fastcall TMainForm::HelpVAClick(TObject *Sender)
{
    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::HelpAboutClick(TObject *Sender)
{
    AboutBox->ShowModal();
//***** remove when not necessary anymore
    InfosBox->ClWLab->Caption = IntToStr(MainForm->ClientWidth);
    InfosBox->ClHLab->Caption = IntToStr(MainForm->ClientHeight);
    InfosBox->WLab->Caption   = IntToStr(MainForm->Width);
    InfosBox->HLab->Caption   = IntToStr(MainForm->Height);
    InfosBox->WStage->Caption = IntToStr(MainForm->ImageStage->Width);
    InfosBox->HStage->Caption = IntToStr(MainForm->ImageStage->Height);

    InfosBox->ShowModal();
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::SelectObject_FRONT(int X, int Y)
{
    //We assume that the object on stage is not selected by default
    ObjectToSelect = -1;

    for (int i = 0; i < ListOfObjects->Count; i++)
    {
        //Extracts the object at the i-th position in the list of objects present
on stage
        StageObject = (TStageObject*) ListOfObjects->Items[i];

        //Is the mouse location inside the clipping rectangle ?
```

```
        if ((X > StageObject->NewPos_FRONT.Left) && (Y > StageObject-
>NewPos_FRONT.Top) && (X < StageObject->NewPos_FRONT.Right) && (Y < StageObject-
>NewPos_FRONT.Bottom))
        {
            //Are the object's white pixels transparent ?
            if (!StageObject->StorageBmp->Transparent)
            {
                //No transparency
                ObjectToSelect = i;     //The object has been selected
            }
            else //Transparency
            {
                //Is the pixel under mouse loc white (white = TransparentColor) or
not ?
                if (StageObject->StorageBmp->Canvas->Pixels[X-StageObject-
>NewPos_FRONT.Left][Y-StageObject->NewPos_FRONT.Top] != clWhite)
                //In TrueColor-32bits, [StageObject->StorageBmp->TransparentColor]
ever holds 50331647 when clWhite = 16777215 ! So it's not possible to compare with
[TransparentColor]
                {
                    //Pixel under mouse loc is not white/transparent
                    ObjectToSelect = i;     //The object has been selected
                }
            }
        }
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::SelectObject_TOP(int X, int Y)
{
    //We assume that the object on stage is not selected by default
    ObjectToSelect = -1;

    for (int i = 0; i < ListOfObjects->Count; i++)
    {
        //Extracts the object at the i-th position in the list of objects present
on stage
        StageObject = (TStageObject*) ListOfObjects->Items[i];

        //Is the mouse location inside the clipping rectangle ?
        if ((X > StageObject->NewPos_TOP.Left) && (Y > StageObject-
>NewPos_TOP.Top) && (X < StageObject->NewPos_TOP.Right) && (Y < StageObject-
>NewPos_TOP.Bottom))
        {
            //Are the object's white pixels transparent ?
            if (!StageObject->StorageBmp->Transparent)
            {
                //No transparency
                ObjectToSelect = i;     //The object has been selected
            }
            else //Transparency
            {
                //Is the pixel under mouse loc white (white = TransparentColor) or
not ?
                if (StageObject->StorageBmp->Canvas->Pixels[X-StageObject-
>NewPos_FRONT.Left][Y-StageObject->NewPos_FRONT.Top] != clWhite)
                //In TrueColor-32bits, [StageObject->StorageBmp->TransparentColor]
ever holds 50331647 when clWhite = 16777215 ! So it's not possible to compare with
[TransparentColor]
                {
                    //Pixel under mouse loc is not white/transparent
                    ObjectToSelect = i;     //The object has been selected
                }
```

```
            }
        }
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::ImageStageMouseDown(TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)
{
    switch (CurTool)
    {
        case tsRectangle :
        case tsCircle :
        case tsLine :
            CanDraw = true;
            Origin  = Point(X,Y);
            MovePt  = Point(X,Y);
            break;
        case tsFreehand :
            CanDrawFreehand = true;

            //Sets pen and brush properties to paint Workbench's surface in white
first
            Workbench->Canvas->Pen->Style   = psSolid;
            Workbench->Canvas->Pen->Mode    = pmCopy;
            Workbench->Canvas->Pen->Color   = clWhite;
            Workbench->Canvas->Brush->Style = bsSolid;
            Workbench->Canvas->Brush->Color = clWhite;

            //Wipes the temporary bmp in white
            Workbench->Canvas->Rectangle(0, 0, Workbench->Width, Workbench-
>Height);

            //Sets pen with the color anf the pen size choosed by the user to draw
off-screen
            Workbench->Canvas->Pen->Color   = ColorDialog->Color;
            Workbench->Canvas->Pen->Width   = CurrentPenSize;

            //Sets pen to draw with on screen
            ImageStage->Canvas->Pen->Style = psSolid;
            ImageStage->Canvas->Pen->Mode  = pmCopy;
            ImageStage->Canvas->Pen->Color = ColorDialog->Color;

            //Places the pen to be ready to draw on both off- and onscreen
            Workbench->Canvas->MoveTo(X, Y);     //Places the pen off-screen on
Workbench
            ImageStage->Canvas->MoveTo(X, Y);    //Places the pen on screen to be
ready to give feedback to the user

            //Sizes a clip.rect. including the pixels colored (freehanded) by the
pencil
            FreeHClipRect = Rect(X, Y, X, Y);
            break;

        case tsMove :
            //We assume that the object on stage is not selected by default and so
can't be moved
            CanMove = false;

            //Calls the appropriate function that determines if an object has been
selected either in FRONT VIEW or in TOP VIEW
            if(ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage -
Front View")
            {//The current view is Front View
```

```
                SelectObject_FRONT(X, Y);

                if (ObjectToSelect > -1)
                {
                    //Extracts the object on which the current tool will be
applied
                    StageObject = (TStageObject*) ListOfObjects-
>Items[ObjectToSelect];

                    CanMove = true;

                    //Calculates the offset between (X,Y) MouseLoc and the left
and top side of the ClipRect
                    XOffset = X - StageObject->NewPos_FRONT.Left;
                    YOffset = Y - StageObject->NewPos_FRONT.Top;

                }//else : select nothing - the user has not clicked on an object
            }
            else //->Panels->Items[0]->Text == " Stage - Top View"
            {
                SelectObject_TOP(X, Y);

                if (ObjectToSelect > -1)
                {
                    //Extracts the object on which the current tool will be
applied
                    StageObject = (TStageObject*) ListOfObjects-
>Items[ObjectToSelect];

                    CanMove = true;

                    //Calculates the offset needed not to place ClipRect's TopLeft
corner in (X,Y) MouseLoc
                    XOffset = X - StageObject->NewPos_TOP.Left;
                    YOffset = Y - ((ImageStage->Height - 1) - StageObject-
>Center_3D.z);    //[YOffset] is the distance between MouseLoc's Y and Center_3D.z

                }//else : select nothing - the user has not clicked on an object
            }
            break;

        case tsTransp :
            //Calls the function that determines if an object has been selected
            SelectObject_FRONT(X, Y);

            if (ObjectToSelect > -1)
            {
                //Extracts the object on which the current tool will be applied
                StageObject = (TStageObject*) ListOfObjects-
>Items[ObjectToSelect];

                //Determines if the selected object must be drawn with
transparency or not
                StageObject->StorageBmp->TransparentColor = clWhite;     //Sets
the color used to apply the transparency effect
                StageObject->StorageBmp->Transparent = !StageObject->StorageBmp-
>Transparent;
                //Calls the method / member function that rebuilds the
StageObjects overlapping the clip.rect. of the modified object
                Rebuild_FRONT(dmObjectClipRect_FrontView);

                //As a safety measure...
                WrappingUp();
```

```
            }//else : select nothing - the user has not clicked on an object
            break;

        case tsDelObj :
            //Calls the appropriate function that determines if an object has been
selected
            if(ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage -
Front View")
            {//The current view is Front View
                SelectObject_FRONT(X, Y);
            }
            else //->Panels->Items[0]->Text == " Stage - Top View"
            {
                SelectObject_TOP(X, Y);
            }

            if (ObjectToSelect > -1)
            {
                //Extracts the object that will be deleted
                StageObject = (TStageObject*) ListOfObjects-
>Items[ObjectToSelect];

                //Frees the memory associated with the object stored at position
[ObjectToSelect]
                delete StageObject;
                StageObject = NULL;
                //Remove the entry in the TList
                ListOfObjects->Delete(ObjectToSelect);

                //The object currently selected has been deleted, so it's no use
to reference it anymore
                ObjectToSelect = -1;

                //As a safety measure...
                //Refreshes also the whole stage
                WrappingUp();

            }//else : select nothing - the user has not clicked on an object
            break;

        //default stand for CASE tsNoTool, that is, when no Tool is selected
        default :  //Do nothing
            break;
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::ImageStageMouseMove(TObject *Sender,
      TShiftState Shift, int X, int Y)
{
    //Displays mouse pointer location on stage - not on screen
    ToolBox->ToolBoxStatusBar->Panels->Items[3]->Text = " X: "+IntToStr(X)+"  ;
Y: "+IntToStr(Y);

    if (CanDraw)
    {
        DrawTempShape(Origin, MovePt);     //Erases previous shape
        MovePt = Point(X,Y);
        DrawTempShape(Origin, MovePt);     //Draws current shape
    };

    if (CanDrawFreehand)
    {
```

```
        //Draws freehanded on both off- and onscreen by following the pointer
location
        Workbench->Canvas->LineTo(X, Y);        //Draws off-screen on Workbench
        ImageStage->Canvas->LineTo(X,Y);        //Draws on screen to give feedback
to the user

        if (X < FreeHClipRect.Left)
            //Enlarges the left side of the clip.rect. surrounding the freehand
drawing
            FreeHClipRect.Left = X;
        if (Y < FreeHClipRect.Top)
            //Enlarges the left side of the clip.rect. surrounding the freehand
drawing
            FreeHClipRect.Top = Y;
        if (X > FreeHClipRect.Right)
            //Enlarges the left side of the clip.rect. surrounding the freehand
drawing
            FreeHClipRect.Right = X;
        if (Y > FreeHClipRect.Bottom)
            //Enlarges the left side of the clip.rect. surrounding the freehand
drawing
            FreeHClipRect.Bottom = Y;
    };

    if (CanMove)    //Bool for the [Move] function
    {
        if (ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage - Front
View")
        {//[Move in FRONT VIEW]
            //Sets previous coordinates to indicate where was the object
            StageObject->PrevPos_FRONT  = StageObject->NewPos_FRONT;

            //Calculates new coordinates to indicate where to draw the object
            StageObject->NewPos_FRONT = GetObjectNewCoord_FRONT(X, Y, XOffset,
YOffset);

            //Merges NewPos_FRONT and PrevPos_FRONT to get a bigger TRect
including both
            MergeRects_FRONT();

            //Erases everything including the object on its prev. pos. and rebuild
everything including the obj. on its. new pos.
            Rebuild_FRONT(dmMergedClipRect);
        }
        else //[Move in TOP VIEW]
        {
            //Sets previous coordinates to indicate where was the TopRect
            StageObject->PrevPos_TOP  = StageObject->NewPos_TOP;

            //Sets new coordinates to indicate where to draw the TopRect
            StageObject->NewPos_TOP = GetObjectNewCoord_TOP(X, Y, XOffset,
YOffset);

            //Merges NewPos_FRONT and PrevPos_FRONT to get a bigger TRect
including both
            MergeRects_TOP();

            //Erases everything including the object on its prev. pos. and rebuild
everything including the obj. on its. new pos.
            Rebuild_TOP(dmMergedClipRect);
        }
    }
}
//---------------------------------------------------------------------------
```

```
void __fastcall TMainForm::ImageStageMouseUp(TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)
{
    if (CanDraw)
    {
        //Stores the final shape
        StoreShape(Origin, Point(X,Y));

        //Calls the method / member function that rebuilds the StageObjects
overlapping the clip.rect. of the modified object
        Rebuild_FRONT(dmObjectClipRect_FrontView);

        //As a safety measure...
        WrappingUp();

        CanDraw = false;
    };

    if (CanDrawFreehand)
    {
        //Allocates a new memory space for that object on stage and returns a
pointer to it
        StageObject = new TStageObject;
        //Gives an ID number to the object
        StageObject->ID = ++LastObjectID;

        //Positions StorageBmp
        StageObject->NewPos_FRONT = FreeHClipRect;

        //The bitmap's height and width properties hold the original size of the
object
        StageObject->StorageBmp->Width  = FreeHClipRect.Right  -
FreeHClipRect.Left;
        StageObject->StorageBmp->Height = FreeHClipRect.Bottom -
FreeHClipRect.Top;

        //When an object is created, its current dimensions are identical to its
original dimensions
        StageObject->CurWidth  = StageObject->StorageBmp->Width;
        StageObject->CurHeight = StageObject->StorageBmp->Height;

        //Records the center in VA's 3D-space
        StageObject->Center_3D.x = StageObject->NewPos_FRONT.Left + (StageObject-
>StorageBmp->Width  / 2);
        StageObject->Center_3D.y = StageObject->NewPos_FRONT.Top  + (StageObject-
>StorageBmp->Height / 2);
        StageObject->Center_3D.z = 0;

        //Stores the final freehand drawing by copying pixels inside FreeHClipRect
onto StorageBmp at position [0,0]
        StageObject->StorageBmp->Canvas->CopyMode = cmSrcCopy;
        StageObject->StorageBmp->Canvas->CopyRect( Rect(0, 0, StageObject-
>StorageBmp->Width, StageObject->StorageBmp->Height), Workbench->Canvas,
FreeHClipRect);

        //Adds StageObject to the list of objects present on stage
        ListOfObjects->Add(StageObject);

        //Sets [ObjectToSelect] to make it reference to the object that has just
been added to the list
        ObjectToSelect = (ListOfObjects->Count) - 1;       //It's the last item
because the list hasn't been sorted since
```

```
        //Calls the method / member function that rebuilds the StageObjects
overlapping the clip.rect. of the modified object
        Rebuild_FRONT(dmObjectClipRect_FrontView);

        //As a safety measure...
        WrappingUp();

        CanDrawFreehand = false;
    };

    if (CanMove)
    {
        CanMove = false;

        if(ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage - Top
View")
        {//The current view is TOP VIEW

            //Sorts the StageObjects stored in the TList according to their Z
coordinate
            SortStageObjects();

            //Stretches or shrinks the object's ClipRect on the screen's surface
for the FRONT VIEW if we move the object respectively forward or backward along
the z coord
            StageObject->PrevPos_FRONT = StageObject->NewPos_FRONT =
DepthRendering();
        }
        else    //Do nothing special - just comment on the reason for which
nothing is done
        {
            //In FRONT VIEW, the [Sorting]        is done after each step of the
[MoveDeep] loop and not here
            //In FRONT VIEW, the [DepthRendering] is done after each step of the
[MoveDeep] loop and not here
        };

        //As a safety measure...
        WrappingUp();
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::SortStageObjects()
{
/*****************************************************************************
*************************/
/* This function sorts the StageObjects in the TList according to their z
coordinate in VA's 3D space.      */
/*      A decreasing order is used to have at the beginning of the list the
objects which are at            */
/*      the back of the stage, and at the end of the list the objects which are at
the front of             */
/*      the stage.  Simply going through the list and displaying the objects one
by one will allow to       */
/*      keep at the front of the stage the objects with the smallest z
coordinates.  Over and above,        */
/*      when an object is created, it is placed at the front of the stage with its
z coord. equal to        */
/*      zero.  So, placing each object newly created at the end of the list avoid
sorting the list to       */
/*      keep it ordered - [TList::Add(void * Item)] places the item at the end of
the TList.                */
```

```
/* Sorting is fairly easy : when [Moving], only one object at a time is not
ordered - that object is        */
/*      the one that has been selected and is referenced by [int ObjectToSelect].
So first delete that      */
/*      entry from the list.  Then loop through the list till you reach the place
where to place that       */
/*      object.  To do it, use [TList::Insert()] or [TList::Add()] according to
the context - but not       */
/*      [TList::Move()] which sometimes puts the object down one place before or
after the one expected.     */
/* Sorting has to be processed at the end of [Move] in TOP VIEW (i.e. OnMouseUp
event) and after every      */
/*      step of the [Move Deep] loop in FRONT VIEW (i.e. OnMouseMove, boolean
CanMoveDeep).                 */
/*
*/
/* The current implementation permits to avoid to be out of bounds.
*/
/*****************************************************************************
**************************/

    //[ptrSelectedStageObject] points to the object that has been selected to be
moved
    TStageObject* ptrSelectedStageObject = (TStageObject*) ListOfObjects-
>Items[ObjectToSelect];

    //Removes the StageObject whose z coordinate has been modified
    ListOfObjects->Delete(ObjectToSelect);

    //The index used to loop through the list. The last value of index [i] when
leaving the loop will give us the position in the list where to place our item
    int i;

    //Determines where (in the list) to replace the StageObject whose z coordinate
has been modified
    for (i=0; i < ListOfObjects->Count; i++)
    {
        //[ptrStageObjectAtCurIndex] points to the object referenced by index [i]
in TList::Items[]
        TStageObject* ptrStageObjectAtCurIndex = (TStageObject*) ListOfObjects-
>Items[i];

        if ( ptrSelectedStageObject->Center_3D.z == ptrStageObjectAtCurIndex-
>Center_3D.z )
        {
            if ( ptrSelectedStageObject->ID < ptrStageObjectAtCurIndex->ID )
            {//We have the new position for [SelectedStageObject]
                break;
            }
        }
        else
        {
            if ( ptrSelectedStageObject->Center_3D.z > ptrStageObjectAtCurIndex-
>Center_3D.z )
            {//We have the new position for [SelectedStageObject]
                break;
            }
        }
    }

    if (i == ListOfObjects->Count)
    {//The StageObject whose z coordinate has been modified is placed at the end
of the list
        ListOfObjects->Add(ptrSelectedStageObject);
```

```
        //Sets [ObjectToSelect] to make it reference to the object that has just
been added to the list
        ObjectToSelect = (ListOfObjects->Count) - 1;        //It's the last item
because TList::Add() places its object (the param.) at the end of the list
    }
    else
    {//The StageObject whose z coordinate has been modified is inserted in the
list between two other items
        ListOfObjects->Insert(i, ptrSelectedStageObject);

        //Sets [ObjectToSelect] to make it reference to the object that has just
been added to the list
        ObjectToSelect = i;     //It's [i] because the object has been inserted at
the i-th position
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::WrappingUp()
{
/*****************************************************************************
**************************/
/* This function makes sure that, if the last operation has resulted in a bug, we
can see                    */
/*      it immediately and not after three or four other operations that didn't
resulted                   */
/*      in any bug at all ! For this, the whole World is rebuild.
*/
/* Also, the position of the ClipRects in the other view (the one not displayed at
the moment) is             */
/*      updated here. It indicates as well that the world has been modified and
needs to be saved           */
/*      as well as it activates/desactivates the functionalities that need it.
*/
/*
*/
/* N.B: [ Workbench->Height -1 ] is used because the 522-nd line is at the 521-st
position                   */
/*****************************************************************************
**************************/

    if (ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage - Front
View")
    {//The current view is Front View

         //As a safety measure, we rebuild the whole world (FRONT VIEW) after
every operation modifying the World
        Rebuild_FRONT(dmWhole);

        if (StageObject != NULL)
        {
            //Updates the position of the ClipRects placed on Stage above the
polygon in TOP VIEW
            //Maintains [PrevPos_TOP] and [NewPos_TOP] according to [Center_3D] -
without using [NewPos_FRONT] because this one could be modified for perspective
effect
            int HalfWidth = StageObject->StorageBmp->Width  / 2;
            StageObject->PrevPos_TOP = StageObject->NewPos_TOP = Rect(
StageObject->Center_3D.x - HalfWidth, ((Workbench->Height-1) - StageObject-
>Center_3D.z) - 6, StageObject->Center_3D.x + HalfWidth, ((Workbench->Height-1) -
StageObject->Center_3D.z) + 6 );
```

```
        }//ELSE : do not de-reference StageObject to avoid an 'Access Violation
Error'
    }
    else //->Panels->Items[0]->Text == " Stage - Top View"
    {
        //As a safety measure, we rebuild the whole world (TOP VIEW) after every
operation modifying the World
        Rebuild_TOP(dmWhole);

        //The ClipRects in FRONT VIEW are updated in [DepthRendering()] for
getting a perspective effect
    }

    //The world has been modified - a new object has been added or something has
been moved / modified
    WorldModified = true;

    //Makes available the appropriate functionalities
    UpdateFunctionsAvailable();
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::UpdateFunctionsAvailable()
{
/*****************************************************************************
*******/
/* This function tests which changes needs to be performed in terms of functions
*/
/*      and views available, buttons lowered, etc. according to the status of
*/
/*      the world and the views.
*/
/*****************************************************************************
*******/

//----- CHANGE IN VIEW ?
    if (ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage - Front
View")
    {//The current view is Front View

        //So disactivate what is related to it and activate what's related to Top
View
        StageTopView->Enabled               = true;
        ToolBox->ToolBtnTopView->Enabled    = true;
        StageFrontView->Enabled             = false;
        ToolBox->ToolBtnFrontView->Enabled  = false;

        //Activate drawing functions like Circle or Line that needs to be drawn by
dragging the mouse
        //      - which can only be done in Front View
        EditInsertObject->Enabled                   = true;
        ToolBox->ToolBtnInsertObject->Enabled       = true;

        EditShapeRectangle->Enabled                 = true;
        ToolBox->ToolBtnRectangle->Enabled          = true;

        EditShapeCircle->Enabled                    = true;
        ToolBox->ToolBtnCircle->Enabled             = true;

        EditShapeLine->Enabled                      = true;
        ToolBox->ToolBtnLine->Enabled               = true;

        EditShapeFreeHandPencil->Enabled            = true;
        ToolBox->ToolBtnFreehandPencil->Enabled     = true;
```

```
        FileSaveWorld->Enabled             = false;
        ToolBox->ToolBtnSaveWorld->Enabled = false;

        FileSaveWorldAs->Enabled             = false;
        ToolBox->ToolBtnSaveWorldAs->Enabled = false;

    //STAGE menu
        StageObjectToWorkRoom->Enabled             = false;
        ToolBox->ToolBtnObjectToWorkRoom->Enabled = false;

        StageClear->Enabled                  = false;
        ToolBox->ToolBtnStageClear->Enabled = false;

        StageCopyAll->Enabled                  = false;
        ToolBox->ToolBtnStageCopyAll->Enabled = false;

    //WORK menu

    //EDIT menu
        EditCleanPixels->Enabled             = false;
        ToolBox->ToolBtnCleanPixels->Enabled = false;

        EditMove->Enabled              = false;
        ToolBox->ToolBtnMove->Enabled  = false;

        EditMoveDeep->Enabled              = false;
        ToolBox->ToolBtnMoveDeep->Enabled  = false;

        EditResizeObject->Enabled              = false;
        ToolBox->ToolBtnResizeObject->Enabled  = false;

        EditFlipHorizontal->Enabled              = false;
        ToolBox->ToolBtnFlipHorizontal->Enabled  = false;

        EditFlipVertical->Enabled              = false;
        ToolBox->ToolBtnFlipVertical->Enabled  = false;

        EditPattern->Enabled             = false;
        ToolBox->ToolBtnPattern->Enabled  = false;

        EditResizePattern->Enabled              = false;
        ToolBox->ToolBtnResizePattern->Enabled  = false;

        EditTransparentObject->Enabled             = false;
        ToolBox->ToolBtnTransparentObject->Enabled = false;

        EditColourColourObject->Enabled        = false;
        ToolBox->ToolBtnColourObject->Enabled  = false;

        EditCutObject->Enabled             = false;
        ToolBox->ToolBtnCutObject->Enabled  = false;

        EditCopyObject->Enabled              = false;
        ToolBox->ToolBtnCopyObject->Enabled  = false;

        EditDeleteObject->Enabled             = false;
        ToolBox->ToolBtnDeleteObject->Enabled = false;

        EditObjectToFloor->Enabled              = false;
        ToolBox->ToolBtnObjectToFloor->Enabled  = false;
    }
}
//---------------------------------------------------------------------------
```

```
        }//ELSE : do not de-reference StageObject to avoid an 'Access Violation
Error'
    }
    else //->Panels->Items[0]->Text == " Stage - Top View"
    {
        //As a safety measure, we rebuild the whole world (TOP VIEW) after every
operation modifying the World
        Rebuild_TOP(dmWhole);

        //The ClipRects in FRONT VIEW are updated in [DepthRendering()] for
getting a perspective effect
    }

    //The world has been modified - a new object has been added or something has
been moved / modified
    WorldModified = true;

    //Makes available the appropriate functionalities
    UpdateFunctionsAvailable();
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::UpdateFunctionsAvailable()
{
/*****************************************************************************
*******/
/* This function tests which changes needs to be performed in terms of functions
*/
/*      and views available, buttons lowered, etc. according to the status of
*/
/*      the world and the views.
*/
/*****************************************************************************
*******/

//----- CHANGE IN VIEW ?
    if (ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage - Front
View")
    {//The current view is Front View

        //So disactivate what is related to it and activate what's related to Top
View
        StageTopView->Enabled               = true;
        ToolBox->ToolBtnTopView->Enabled    = true;
        StageFrontView->Enabled             = false;
        ToolBox->ToolBtnFrontView->Enabled  = false;

        //Activate drawing functions like Circle or Line that needs to be drawn by
dragging the mouse
        //      - which can only be done in Front View
        EditInsertObject->Enabled                   = true;
        ToolBox->ToolBtnInsertObject->Enabled       = true;

        EditShapeRectangle->Enabled                 = true;
        ToolBox->ToolBtnRectangle->Enabled          = true;

        EditShapeCircle->Enabled                    = true;
        ToolBox->ToolBtnCircle->Enabled             = true;

        EditShapeLine->Enabled                      = true;
        ToolBox->ToolBtnLine->Enabled               = true;

        EditShapeFreeHandPencil->Enabled            = true;
        ToolBox->ToolBtnFreehandPencil->Enabled     = true;
```

```
        EditPenSize->Enabled                          = true;
        ToolBox->ToolBtnPenSize->Enabled              = true;

        //Restores the hint property of [Move] and [Move Deep] since they have a
different behavior from the one they've got in TOP VIEW
        ToolBox->ToolBtnMove->Hint     = "Move an object on the vertical plane";
        ToolBox->ToolBtnMoveDeep->Hint = "Move an object on the horizontal plane";

        //[TransparentObject] is enabled if we are in TOP VIEW with at least 1
object on the stage
    }
    else
    {//The current view is Top View

        //So disactivate what is related to it and activate what's related to
Front View
        StageTopView->Enabled               = false;
        ToolBox->ToolBtnTopView->Enabled    = false;
        StageFrontView->Enabled             = true;
        ToolBox->ToolBtnFrontView->Enabled  = true;

        //Disactivate drawing functions like Circle or Line that needs to be drawn
by dragging the mouse
        //       - which can only be done in Front View
        EditInsertObject->Enabled                     = false;
        ToolBox->ToolBtnInsertObject->Enabled         = false;

        EditShapeRectangle->Enabled                   = false;
        ToolBox->ToolBtnRectangle->Enabled            = false;

        EditShapeCircle->Enabled                      = false;
        ToolBox->ToolBtnCircle->Enabled               = false;

        EditShapeLine->Enabled                        = false;
        ToolBox->ToolBtnLine->Enabled                 = false;

        EditShapeFreeHandPencil->Enabled              = false;
        ToolBox->ToolBtnFreehandPencil->Enabled       = false;

        EditPenSize->Enabled                          = false;
        ToolBox->ToolBtnPenSize->Enabled              = false;

        //Changes the hint property of [Move] and [Move Deep] since they behave
differently and are the same functions in TOP VIEW
        ToolBox->ToolBtnMove->Hint     = "Move an object backwards and forwards";
        ToolBox->ToolBtnMoveDeep->Hint = "Move an object backwards and forwards";

        //[TransparentObject] is not enabled in TOP VIEW.
    }

//----- AT LEAST ONE OBJECT ON THE STAGE ?
    if (ListOfObjects->Count != 0)
    {// >= 1 objects on the stage
    //FILE menu
        FileSaveWorld->Enabled              = true;
        ToolBox->ToolBtnSaveWorld->Enabled = true;

        FileSaveWorldAs->Enabled             = true;
        ToolBox->ToolBtnSaveWorldAs->Enabled = true;

    //STAGE menu
        StageObjectToWorkRoom->Enabled             = true;
        ToolBox->ToolBtnObjectToWorkRoom->Enabled = true;
```

```
        StageClear->Enabled                  = true;
        ToolBox->ToolBtnStageClear->Enabled = true;

        StageCopyAll->Enabled                  = true;
        ToolBox->ToolBtnStageCopyAll->Enabled = true;

    //WORK menu

    //EDIT menu
        EditCleanPixels->Enabled              = true;
        ToolBox->ToolBtnCleanPixels->Enabled = true;

        EditMove->Enabled               = true;
        ToolBox->ToolBtnMove->Enabled   = true;

        EditMoveDeep->Enabled               = true;
        ToolBox->ToolBtnMoveDeep->Enabled   = true;

        EditResizeObject->Enabled               = true;
        ToolBox->ToolBtnResizeObject->Enabled   = true;

        EditFlipHorizontal->Enabled               = true;
        ToolBox->ToolBtnFlipHorizontal->Enabled   = true;

        EditFlipVertical->Enabled               = true;
        ToolBox->ToolBtnFlipVertical->Enabled   = true;

        EditPattern->Enabled               = true;
        ToolBox->ToolBtnPattern->Enabled   = true;

        EditResizePattern->Enabled               = true;
        ToolBox->ToolBtnResizePattern->Enabled   = true;

        if(ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage - Front
View")
        {//The current view is Front View
            EditTransparentObject->Enabled              = true;
            ToolBox->ToolBtnTransparentObject->Enabled = true;
        }
        else //->Panels->Items[0]->Text == " Stage - Top View"
        {
            EditTransparentObject->Enabled              = false;
            ToolBox->ToolBtnTransparentObject->Enabled = false;
        }

        EditColourColourObject->Enabled        = true;
        ToolBox->ToolBtnColourObject->Enabled  = true;

        EditCutObject->Enabled               = true;
        ToolBox->ToolBtnCutObject->Enabled   = true;

        EditCopyObject->Enabled               = true;
        ToolBox->ToolBtnCopyObject->Enabled   = true;

        EditDeleteObject->Enabled              = true;
        ToolBox->ToolBtnDeleteObject->Enabled = true;

        EditObjectToFloor->Enabled               = true;
        ToolBox->ToolBtnObjectToFloor->Enabled   = true;
    }
    else //There are no objects on the Stage
    {
    //FILE menu
```

```
        FileSaveWorld->Enabled             = false;
        ToolBox->ToolBtnSaveWorld->Enabled = false;

        FileSaveWorldAs->Enabled             = false;
        ToolBox->ToolBtnSaveWorldAs->Enabled = false;

    //STAGE menu
        StageObjectToWorkRoom->Enabled             = false;
        ToolBox->ToolBtnObjectToWorkRoom->Enabled = false;

        StageClear->Enabled                  = false;
        ToolBox->ToolBtnStageClear->Enabled = false;

        StageCopyAll->Enabled                  = false;
        ToolBox->ToolBtnStageCopyAll->Enabled = false;

    //WORK menu

    //EDIT menu
        EditCleanPixels->Enabled             = false;
        ToolBox->ToolBtnCleanPixels->Enabled = false;

        EditMove->Enabled              = false;
        ToolBox->ToolBtnMove->Enabled  = false;

        EditMoveDeep->Enabled              = false;
        ToolBox->ToolBtnMoveDeep->Enabled  = false;

        EditResizeObject->Enabled              = false;
        ToolBox->ToolBtnResizeObject->Enabled  = false;

        EditFlipHorizontal->Enabled              = false;
        ToolBox->ToolBtnFlipHorizontal->Enabled  = false;

        EditFlipVertical->Enabled              = false;
        ToolBox->ToolBtnFlipVertical->Enabled  = false;

        EditPattern->Enabled              = false;
        ToolBox->ToolBtnPattern->Enabled  = false;

        EditResizePattern->Enabled              = false;
        ToolBox->ToolBtnResizePattern->Enabled  = false;

        EditTransparentObject->Enabled             = false;
        ToolBox->ToolBtnTransparentObject->Enabled = false;

        EditColourColourObject->Enabled        = false;
        ToolBox->ToolBtnColourObject->Enabled  = false;

        EditCutObject->Enabled              = false;
        ToolBox->ToolBtnCutObject->Enabled  = false;

        EditCopyObject->Enabled              = false;
        ToolBox->ToolBtnCopyObject->Enabled  = false;

        EditDeleteObject->Enabled             = false;
        ToolBox->ToolBtnDeleteObject->Enabled = false;

        EditObjectToFloor->Enabled              = false;
        ToolBox->ToolBtnObjectToFloor->Enabled  = false;
    }
}
//---------------------------------------------------------------------------
```

```
void __fastcall TMainForm::Rebuild_FRONT(TDisplayMode DisplayMode)
{
/*****************************************************************************
*****************************/
/* Rebuild_FRONT draws first a blank stage, then a wireframe.  Eventually, it
draws all the objects             */
/*      listed as being present on a stage.  When VA is launched, if the user
didn't 2x-click on a VA file,     */
/*      the list contains 0 objects and only a blank stage + a wireframe are drawn
- this is used                 */
/*      to prepare a new stage set. Everything is drawn off-screen and finally
displayed on screen              */
/*****************************************************************************
*****************************/
//***** Should have a parameter to tell which stage is concerned

    CreateBlankStage_OffScr();
    S_H_Wireframe_OffScr();
    //***** Paint the floor if necess.
    PaintObjects_OffScr(DisplayMode);

    //Copies Workbench or a part of it on the screen - detects if the object is
being moved
    Display_OnScr(DisplayMode);
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::CreateBlankStage_OffScr()
{
/*****************************************************************/
/* Just wipes the stage with the color used for the Stage       */
/*****************************************************************/

    //Sets pen and brush parameters to draw the blank stage
    Workbench->Canvas->Pen->Style   = psSolid;
    Workbench->Canvas->Pen->Mode    = pmCopy;
    Workbench->Canvas->Pen->Color   = StageColor;
    Workbench->Canvas->Brush->Style = bsSolid;
    Workbench->Canvas->Brush->Color = StageColor;

    //Draws a rectangle symbolizing the blank stage
    Workbench->Canvas->Rectangle(0, 0, ImageStage->Width, ImageStage->Height);
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::S_H_Wireframe_OffScr()
{
/*****************************************************************************
**********************/
/* Shows or hides the wires on the stage according to the WireframeMode, with the
illusion of depth    */
/*****************************************************************************
**********************/

    switch (WireframeMode)
    {
        case (wfShow) :
            //Sets drawing properties
            Workbench->Canvas->Pen->Style   = psSolid;
            Workbench->Canvas->Pen->Mode    = pmCopy;
            Workbench->Canvas->Pen->Color   = clLime;
            Workbench->Canvas->Pen->Width   = 1;
            Workbench->Canvas->Brush->Style = bsClear;
            break;
```

```
        case (wfHide) :
            //Sets drawing properties
            Workbench->Canvas->Pen->Style   = psSolid;
            Workbench->Canvas->Pen->Mode    = pmCopy;
            Workbench->Canvas->Pen->Color   = MainForm->StageColor;
            Workbench->Canvas->Pen->Width   = 1;
            Workbench->Canvas->Brush->Style = bsClear;
            break;
    };

//PART I : Determines the location of the wireframe when projected on the screen's
surface (for perspective)

    //Sides of the projection on the screen's surface of the wireframe's rectangle
placed at the back
    int WF_LeftSide, WF_RightSide, WF_BottomSide, WF_TopSide;

    //Left side of the Wireframe with perspective first, right side then (read
comments there -->)
    WF_LeftSide   = ConvertLocation_3Dto2D( ImageStage->Width /2 , 0
, ImageStage->Height -1);      //In TOP VIEW, the Left-Top  corner is at (0
, 0), that is (0                    , ImageStage->Height -1) in VA's 3D-space
    WF_RightSide  = ConvertLocation_3Dto2D( ImageStage->Width /2 , ImageStage-
>Width -1, ImageStage->Height -1);      //In TOP VIEW, the Right-Top corner is at
(ImageStage->Width -1, 0), that is (ImageStage->Width -1, ImageStage->Height -1)
in VA's 3D-space

    //Bottom side of the Wireframe with perspective first, top side then
    //You've got to look at VA's 3D-space from the right side and erect it so that
the top is on the left of the screen and the bottom on the right.
    WF_TopSide    = ConvertLocation_3Dto2D( ImageStage->Height /2 , 0
, ImageStage->Height -1);
    WF_BottomSide = ConvertLocation_3Dto2D( ImageStage->Height /2 , ImageStage-
>Height -1, ImageStage->Height -1);

//PART II : Draws 1 rectangle and 4 lines
    Workbench->Canvas->Rectangle(WF_LeftSide, WF_TopSide, WF_RightSide,
WF_BottomSide);
    Workbench->Canvas->MoveTo(0, 0);
    Workbench->Canvas->LineTo(WF_LeftSide, WF_TopSide);
    Workbench->Canvas->MoveTo(Workbench->Width, 0);
    Workbench->Canvas->LineTo(WF_RightSide, WF_TopSide);
    Workbench->Canvas->MoveTo(0, Workbench->Height);
    Workbench->Canvas->LineTo(WF_LeftSide, WF_BottomSide);
    Workbench->Canvas->MoveTo(Workbench->Width, Workbench->Height);
    Workbench->Canvas->LineTo(WF_RightSide, WF_BottomSide);
}
//---------------------------------------------------------------------------

void  __fastcall TMainForm::PaintObjects_OffScr(TDisplayMode DisplayMode)
{
/*****************************************************************************
***************/
/* This method loops through the list of objects present on the stage and draws
their bitmap    */
/*      (embedded in StageObject under the name StorageBmp) on Workbench (off-
screen bitmap)    */
/*      only if they are inside a certain ClipRect that is determined by
DisplayMode.          */
/*****************************************************************************
***************/

    //Points to the StageObject currently considered - we need to know which
ClipRect to consider in the [case dmObjectClipRect_FrontView].
```

```
    TStageObject* CurConsObject;

    for(int i = 0; i < ListOfObjects->Count; i++)
    {
        //Extracts the object at the i-th position in the list of objects present
on stage
        StageObject = (TStageObject*) ListOfObjects->Items[i];

        switch (DisplayMode)
        {
            case dmWhole :
                //Displays all objects present on the stage
                Workbench->Canvas->StretchDraw(StageObject->NewPos_FRONT,
StageObject->StorageBmp);
                break;

            case dmObjectClipRect_FrontView :
                //Displays only objects inside the StageObject's clipping
rectangle currently considered to apply an effect on the object

                //Extracts the StageObject currently considered
                CurConsObject = (TStageObject*) ListOfObjects-
>Items[ObjectToSelect];

                if ( IntersectRects(StageObject->NewPos_FRONT, CurConsObject-
>NewPos_FRONT) )
                {
                    //The object is inside the StageObject's clipping rectangle
currently considered, so paint it onto Workbench
                    Workbench->Canvas->StretchDraw(StageObject->NewPos_FRONT,
StageObject->StorageBmp);

                }//ELSE : do not display the object at the i-th position in the
list because the object is not inside the StageObject's clipping rectangle
currently considered
                break;

            case dmMergedClipRect :
                //Displays only objects inside the MergedRectangle used to move an
object
                if ( IntersectRects(StageObject->NewPos_FRONT, MergedRectangle) )
                {
                    //The object is inside MergedRectangle, so paint it onto
Workbench
                    Workbench->Canvas->StretchDraw(StageObject->NewPos_FRONT,
StageObject->StorageBmp);

                }//ELSE : do not display the object at the i-th position in the
list because the object is not inside MergedRectangle
                break;
        }
    }

    if (ObjectToSelect > -1)
    {
        //Restore the object which has been selected and is used for the moment
        StageObject = (TStageObject*) ListOfObjects->Items[ObjectToSelect];

    }//else : select nothing - the user has not clicked on an object
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::Display_OnScr(TDisplayMode DisplayMode)
{
```

```
/*****************************************************************************
**********************/
/* This function determines which part of the off-screen bitmap has to be
displayed                    */
/*      that is to say :    dmWhole                     --> the whole off-screen
is displayed          */
/*                          dmObjectClipRect_FrontView  --> the object only, in
its Front View         */
/*                          dmObjectClipRect_TopView    --> the object only, in
its Top View           */
/*                          dmMergedClipRect            --> the clip. rect
including the object        */
/*                                                          both on its previous
and new positions     */
/*****************************************************************************
**********************/

    //Sets mode used with CopyRect
    ImageStage->Canvas->CopyMode = cmSrcCopy;

    //Displays Workbench on screen according to the DisplayMode
    TRect AreaToDisplay;
    switch (DisplayMode)
    {
        case dmWhole :
            AreaToDisplay = Rect(0, 0, ImageStage->Width, ImageStage->Height);
            break;
        case dmObjectClipRect_FrontView :
            AreaToDisplay = StageObject->NewPos_FRONT;
            break;
        case dmObjectClipRect_TopView :
            //In TOP VIEW, there's a ClipRect surrounding the object's polygon
when it is angled
            AreaToDisplay = StageObject->NewPos_TOP;
            break;
        case dmMergedClipRect :
            AreaToDisplay = MergedRectangle;
            break;
    };
    ImageStage->Canvas->CopyRect(AreaToDisplay , Workbench->Canvas,
AreaToDisplay);
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::Rebuild_TOP(TDisplayMode DisplayMode)
{
/*****************************************************************************
*****************************/
/* Rebuild_TOP draws first a blank floor of a stage, then it draws a polygon to be
able to select an object     */
/*      listed as being present on a stage.  Everything is drawn off-screen and
finally displayed on screen     */
/*****************************************************************************
*****************************/
//***** Should have a parameter to tell which stage is concerned

    CreateBlankFloor_OffScr();
    DrawTopRects_OffScr(DisplayMode);
    //Copies Workbench or a part of it on the screen - detects if the object is
being moved
    Display_OnScr(DisplayMode);
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::CreateBlankFloor_OffScr()
{
```

```
/*******************************************************/
/* Just wipes the stage's floor with its own color     */
/*******************************************************/

    //Sets pen and brush parameters to draw the floor
    Workbench->Canvas->Pen->Style   = psSolid;
    Workbench->Canvas->Pen->Mode    = pmCopy;
    Workbench->Canvas->Pen->Color   = FloorColor;
    Workbench->Canvas->Brush->Style = bsSolid;
    Workbench->Canvas->Brush->Color = FloorColor;

    //Draws a rectangle symbolizing the blank stage
    Workbench->Canvas->Rectangle(0, 0, ImageStage->Width, ImageStage->Height);
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::DrawTopRects_OffScr(TDisplayMode DisplayMode)
{
/*****************************************************************************
**************/
/* This method loops through the list of objects present on the stage and draws a
polygon thus  */
/*      giving a virtual depth to the object in order to grab and move it.
Everything is       */
/*      drawn on Workbench (off-screen bitmap) only if they are inside a certain
ClipRect       */
/*      that is determined by DisplayMode.  Note that it is especially useful when
the object   */
/*      is being angled.
*/
/*****************************************************************************
**************/

    //Sets pen and brush properties to paint with
    Workbench->Canvas->Pen->Style   = psSolid;
    Workbench->Canvas->Pen->Mode    = pmCopy;
    Workbench->Canvas->Pen->Color   = clAqua;
    Workbench->Canvas->Pen->Width   = 1;
    Workbench->Canvas->Brush->Style = bsSolid;
    Workbench->Canvas->Brush->Color = clGray;

    //Points to the StageObject currently considered - we need to know which
ClipRect to consider in the [case dmObjectClipRect_FrontView].
    TStageObject* CurConsObject;

    for(int i = 0; i < ListOfObjects->Count; i++)
    {
        //Extracts the object at the i-th position in the list of objects present
on stage
        StageObject = (TStageObject*) ListOfObjects->Items[i];

        switch (DisplayMode)
        {
            case dmWhole :
                //Displays all objects present on the stage by drawing a polygon
to be able to see the objects from TOP VIEW
                //***** Later on, use the method [POLYGON]
                Workbench->Canvas->Rectangle(StageObject->NewPos_TOP.Left,
StageObject->NewPos_TOP.Top, StageObject->NewPos_TOP.Right, StageObject-
>NewPos_TOP.Bottom);
                break;

            case dmObjectClipRect_TopView :
                //Displays only object's polygons inside the ClipRect of the
StageObject being angled (remember we are in TOP VIEW)
```

```
                //Extracts the StageObject currently considered (being angled)
                CurConsObject = (TStageObject*) ListOfObjects-
>Items[ObjectToSelect];

                if ( IntersectRects(StageObject->NewPos_TOP, CurConsObject-
>NewPos_TOP) )
                {
                    //The object is inside the StageObject's clipping rectangle
currently angled, so draw its polygon onto Workbench
                    //***** Later on, use the method [POLYGON]
                    Workbench->Canvas->Rectangle(StageObject->NewPos_TOP.Left,
StageObject->NewPos_TOP.Top, StageObject->NewPos_TOP.Right, StageObject-
>NewPos_TOP.Bottom);

                }//ELSE : do not display the object at the i-th position in the
list because the object is not inside the StageObject's clipping rectangle
currently considered
                break;

            case dmMergedClipRect :
                //Displays only objects inside the MergedRectangle used to move an
object's polygon
                if ( IntersectRects(StageObject->NewPos_TOP, MergedRectangle) )
                {
                    //The object is inside MergedRectangle, so draw its polygon
onto Workbench
                    //***** Later on, use the method [POLYGON]
                    Workbench->Canvas->Rectangle(StageObject->NewPos_TOP.Left,
StageObject->NewPos_TOP.Top, StageObject->NewPos_TOP.Right, StageObject-
>NewPos_TOP.Bottom);

                }//ELSE : do not display the object at the i-th position in the
list because the object is not inside MergedRectangle
                break;
        }
    }

    if (ObjectToSelect > -1)
    {
        //Restore the object which has been selected and is used for the moment
        StageObject = (TStageObject*) ListOfObjects->Items[ObjectToSelect];

    }//else : select nothing - the user has not clicked on an object
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::DrawTempShape(POINT MouseLocOnClick, POINT
MouseLocOnRelease)
{
/*****************************************************************************/
/* This method gives the user the ability to draw a shape to the size  wanted   */
/*****************************************************************************/

    //Sets pen and brush parameters to draw with
    ImageStage->Canvas->Pen->Style   = psSolid;
    ImageStage->Canvas->Pen->Mode    = pmNotXor;
    ImageStage->Canvas->Pen->Color   = ColorDialog->Color;      //MainForm's data
member used
    ImageStage->Canvas->Pen->Width   = 1;
    ImageStage->Canvas->Brush->Style = bsSolid;
    ImageStage->Canvas->Brush->Color = ColorDialog->Color;      //MainForm's data
member used
```

```
    //Draws according to the type of the tool selected
    switch (CurTool)
    {
        case tsLine :
            ImageStage->Canvas->Pen->Width = CurrentPenSize;
            ImageStage->Canvas->MoveTo(MouseLocOnClick.x, MouseLocOnClick.y);
            ImageStage->Canvas->LineTo(MouseLocOnRelease.x, MouseLocOnRelease.y);
            break;
        case tsRectangle :
            ImageStage->Canvas->Rectangle(MouseLocOnClick.x, MouseLocOnClick.y,
MouseLocOnRelease.x, MouseLocOnRelease.y);
            break;
        case tsCircle :     //Makes square boundaries to get a circle
            //Local variables to get a circle
            int TempWidth, TempHeight;
            TempWidth  = max(MouseLocOnClick.x, MouseLocOnRelease.x) -
min(MouseLocOnClick.x, MouseLocOnRelease.x);
            TempHeight = max(MouseLocOnClick.y, MouseLocOnRelease.y) -
min(MouseLocOnClick.y, MouseLocOnRelease.y);
            //Does it start on the left or right ?
            if (MouseLocOnRelease.x > MouseLocOnClick.x)
            {//Starts on the left

                //Does it start on the top or bottom ?
                if (MouseLocOnRelease.y > MouseLocOnClick.y)
                {//Starts on the top

                    //Normal case : start rendering from the top left corner
                    ImageStage->Canvas->Ellipse(MouseLocOnClick.x,
MouseLocOnClick.y, MouseLocOnClick.x + min(TempWidth, TempHeight),
MouseLocOnClick.y + min(TempWidth, TempHeight));
                }
                else
                {//starts on the bottom

                    //Start rendering from the bottom left corner
                    ImageStage->Canvas->Ellipse(MouseLocOnClick.x,
MouseLocOnClick.y, MouseLocOnClick.x + min(TempWidth, TempHeight),
MouseLocOnClick.y - min(TempWidth, TempHeight));
                }
            }
            else
            {//Starts on the right

                //Does it start on the top or bottom ?
                if (MouseLocOnRelease.y > MouseLocOnClick.y)
                {//Starts on the top

                    //Start rendering from the top right corner
                    ImageStage->Canvas->Ellipse(MouseLocOnClick.x,
MouseLocOnClick.y, MouseLocOnClick.x - min(TempWidth, TempHeight),
MouseLocOnClick.y + min(TempWidth, TempHeight));
                }
                else
                {//starts on the bottom

                    //Start rendering from the bottom right corner
                    ImageStage->Canvas->Ellipse(MouseLocOnClick.x,
MouseLocOnClick.y, MouseLocOnClick.x - min(TempWidth, TempHeight),
MouseLocOnClick.y - min(TempWidth, TempHeight));
                }
            }
            break;
        case tsEraser :
```

```
            if (MessageDlg("Under construction - Not implemented yet !",
mtWarning,
                TMsgDlgButtons() << mbOK, 0) == mrOk)
                //does nothing
                ;
            break;
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StoreShape(POINT MouseLocOnClick, POINT
MouseLocOnRelease)
{
/*****************************************************************************
***********/
/* StoreShape creates temporarily a bitmap of the same size as the stage, wipes it
*/
/*      in white and then draws the shape on it at the same position as on the
screen.      */
/*      Finally, the smallest clip. rect. including the shape is copied onto
StorageBmp.    */
/*      To get a circle, it uses Temp- variables to draw an ellipse in square
boundaries    */
/*****************************************************************************
***********/

// PART I : Paints the temporary bitmap in white

    //Creates a temporary bmp
    Graphics::TBitmap* TempBmp = new Graphics::TBitmap;

    //Allocates a new memory space for that object on stage and returns a pointer
to it
    StageObject = new TStageObject;
    //Gives an ID number to the object
    StageObject->ID = ++LastObjectID;

    //Gives StorageBmp the size of ImageStage
    TempBmp->Width  = ImageStage->Width;
    TempBmp->Height = ImageStage->Height;

    //Sets pen and brush properties to paint the bitmap's surface in white first
    TempBmp->Canvas->Pen->Style   = psSolid;
    TempBmp->Canvas->Pen->Mode    = pmCopy;
    TempBmp->Canvas->Pen->Color   = clWhite;
    TempBmp->Canvas->Brush->Style = bsSolid;
    TempBmp->Canvas->Brush->Color = clWhite;
    //Wipes in white
    TempBmp->Canvas->Rectangle(0, 0, TempBmp->Width, TempBmp->Height);

// PART II : Stores (draws) the shape in the temporary bitmap

    //Sets pen and brush color to draw with on TempBmp
    TempBmp->Canvas->Pen->Color   = ColorDialog->Color;
    TempBmp->Canvas->Brush->Color = ColorDialog->Color;
    TempBmp->Canvas->Pen->Width   = 1;

    //Local variables recording the actual size/position of the shape
    int TempShapeWidth, TempShapeHeight, TempShapeLeft, TempShapeTop,
TempShapeRight, TempShapeBottom;

    //At this moment, the size is the one of the rect. ending on last mouse loc.
With a circle, it'll be reduced to the actual size of it.
    TempShapeWidth  = abs(MouseLocOnRelease.x - MouseLocOnClick.x);
```

```
    TempShapeHeight = abs(MouseLocOnRelease.y - MouseLocOnClick.y);

    //At this moment, the position is the one of the rect. ending on last mouse
loc.  With a circle, it'll be set to the actual pos. of it.
    TempShapeLeft   = min(MouseLocOnClick.x, MouseLocOnRelease.x);
    TempShapeTop    = min(MouseLocOnClick.y, MouseLocOnRelease.y);
    TempShapeRight  = max(MouseLocOnClick.x, MouseLocOnRelease.x);
    TempShapeBottom = max(MouseLocOnClick.y, MouseLocOnRelease.y);

    //Draws according to the type of the tool selected
    switch (CurTool)
    {
        case tsLine :
            TempBmp->Canvas->Pen->Width = CurrentPenSize;
            TempBmp->Canvas->MoveTo(MouseLocOnClick.x, MouseLocOnClick.y);
            TempBmp->Canvas->LineTo(MouseLocOnRelease.x, MouseLocOnRelease.y);
            break;
        case tsRectangle :
            TempBmp->Canvas->Rectangle(MouseLocOnClick.x, MouseLocOnClick.y,
MouseLocOnRelease.x, MouseLocOnRelease.y);
            break;
        case tsCircle :
            //Does it start on the left or right ?
            if (MouseLocOnRelease.x > MouseLocOnClick.x)
            {//Starts on the left

                //Does it start on the top or bottom ?
                if (MouseLocOnRelease.y > MouseLocOnClick.y)
                {//Starts on the top

                    //Normal case : starts rendering from the top left corner
                    TempBmp->Canvas->Ellipse(MouseLocOnClick.x, MouseLocOnClick.y,
MouseLocOnClick.x + min(TempShapeWidth, TempShapeHeight), MouseLocOnClick.y +
min(TempShapeWidth, TempShapeHeight));
                    //Records the actual position of the circle, not the one of
the clip.rect. determined by MouseLocOnClick and MouseLocOnRelease
                    TempShapeLeft   = MouseLocOnClick.x;
                    TempShapeTop    = MouseLocOnClick.y;
                    TempShapeRight  = MouseLocOnClick.x + min(TempShapeWidth,
TempShapeHeight);
                    TempShapeBottom = MouseLocOnClick.y + min(TempShapeWidth,
TempShapeHeight);

                }
                else
                {//starts on the bottom

                    //Starts rendering from the bottom left corner
                    TempBmp->Canvas->Ellipse(MouseLocOnClick.x, MouseLocOnClick.y,
MouseLocOnClick.x + min(TempShapeWidth, TempShapeHeight), MouseLocOnClick.y -
min(TempShapeWidth, TempShapeHeight));
                    //Records the actual position of the circle, not the one of
the clip.rect. determined by MouseLocOnClick to MouseLocOnRelease
                    TempShapeLeft   = MouseLocOnClick.x;
                    TempShapeTop    = MouseLocOnClick.y - min(TempShapeWidth,
TempShapeHeight);
                    TempShapeRight  = MouseLocOnClick.x + min(TempShapeWidth,
TempShapeHeight);
                    TempShapeBottom = MouseLocOnClick.y;
                }
            }
            else
            {//Starts on the right
```

```
                //Does it start on the top or bottom ?
                if (MouseLocOnRelease.y > MouseLocOnClick.y)
                {//Starts on the top

                    //Starts rendering from the top right corner and records the
actual size of the circle
                    TempBmp->Canvas->Ellipse(MouseLocOnClick.x, MouseLocOnClick.y,
MouseLocOnClick.x - min(TempShapeWidth, TempShapeHeight), MouseLocOnClick.y +
min(TempShapeWidth, TempShapeHeight));
                    //Records the actual position of the circle, not the one of
the clip.rect. determined by MouseLocOnClick to MouseLocOnRelease
                    TempShapeLeft   = MouseLocOnClick.x - min(TempShapeWidth,
TempShapeHeight);
                    TempShapeTop    = MouseLocOnClick.y;
                    TempShapeRight  = MouseLocOnClick.x;
                    TempShapeBottom = MouseLocOnClick.y + min(TempShapeWidth,
TempShapeHeight);
                }
                else
                {//starts on the bottom

                    //Starts rendering from the bottom right corner and records
the actual size of the circle
                    TempBmp->Canvas->Ellipse(MouseLocOnClick.x, MouseLocOnClick.y,
MouseLocOnClick.x - min(TempShapeWidth, TempShapeHeight), MouseLocOnClick.y -
min(TempShapeWidth, TempShapeHeight));
                    //Records the actual position of the circle, not the one of
the clip.rect. determined by MouseLocOnClick to MouseLocOnRelease
                    TempShapeLeft   = MouseLocOnClick.x - min(TempShapeWidth,
TempShapeHeight);
                    TempShapeTop    = MouseLocOnClick.y - min(TempShapeWidth,
TempShapeHeight);
                    TempShapeRight  = MouseLocOnClick.x;
                    TempShapeBottom = MouseLocOnClick.y;
                }
            }
            //Records the actual size of the circle rather than measuring the
distance to the last mouse loc. given by MouseLocOnRelease
            TempShapeWidth = TempShapeHeight = min(TempShapeWidth,
TempShapeHeight);
            break;
    }

// PART III : Copies the smallest clip. rect. including the shape and records
//            the position of the object on screen in the object's data structure.

    //The bitmap's height and width properties hold the original size of the
object
    StageObject->StorageBmp->Width  = TempShapeWidth;
    StageObject->StorageBmp->Height = TempShapeHeight;

    //When an object is created, its current dimensions are identical to its
original dimensions
    StageObject->CurWidth  = StageObject->StorageBmp->Width;
    StageObject->CurHeight = StageObject->StorageBmp->Height;

    //Recordsd its position on screen
    StageObject->NewPos_FRONT.Left   = TempShapeLeft;
    StageObject->NewPos_FRONT.Top    = TempShapeTop;
    StageObject->NewPos_FRONT.Right  = TempShapeRight;
    StageObject->NewPos_FRONT.Bottom = TempShapeBottom;

    //Records the center in VA's 3D-space
```

```
    StageObject->Center_3D.x = StageObject->NewPos_FRONT.Left + (StageObject-
>StorageBmp->Width  / 2);
    StageObject->Center_3D.y = StageObject->NewPos_FRONT.Top  + (StageObject-
>StorageBmp->Height / 2);
    StageObject->Center_3D.z = 0;

    //Copies pixels at NewPos_FRONT in TempBmp onto StorageBmp at position [0,0]
    StageObject->StorageBmp->Canvas->CopyMode = cmSrcCopy;
    StageObject->StorageBmp->Canvas->CopyRect( Rect(0, 0, StageObject->StorageBmp-
>Width, StageObject->StorageBmp->Height), TempBmp->Canvas, StageObject-
>NewPos_FRONT);
    delete TempBmp;

    //Adds StageObject to the list of objects present on stage
    ListOfObjects->Add(StageObject);

    //Sets [ObjectToSelect] to make it reference to the object that has just been
added to the list
    ObjectToSelect = (ListOfObjects->Count) - 1;       //It's the last item
because the list hasn't been sorted since
}
//---------------------------------------------------------------------------

__fastcall TStageObject::TStageObject()     //Constructor
{
    StorageBmp = new Graphics::TBitmap;
    StorageBmp->TransparentColor = clWhite;     //Sets the color used to apply the
transparency effect
    StorageBmp->Transparent = true;             //Any object's white pixels are
not visible when the object is created
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::MergeRects_FRONT()
{
/*************************************************************************/
/* This member function (method) takes two rects and records a Merged rect    */
/*      that includes both preceding - used for FRONT VIEW                    */
/* For the moment, we don't pass any rects as parameters, we use data members */
/*      even though this function could be more general - which is not useful */
/*      as far as VA is not concerned.                                        */
/*************************************************************************/

    if(ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage - Front View")
    {
        MergedRectangle.Left     = min(StageObject->PrevPos_FRONT.Left,
StageObject->NewPos_FRONT.Left);
        MergedRectangle.Top      = min(StageObject->PrevPos_FRONT.Top,
StageObject->NewPos_FRONT.Top);
        MergedRectangle.Right    = max(StageObject->PrevPos_FRONT.Right,
StageObject->NewPos_FRONT.Right);
        MergedRectangle.Bottom   = max(StageObject->PrevPos_FRONT.Bottom,
StageObject->NewPos_FRONT.Bottom);
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::MergeRects_TOP()
{
/*************************************************************************/
/* This member function (method) takes two rects and records a Merged rect    */
/*      that includes both preceding - used for TOP VIEW                      */
/* For the moment, we don't pass any rects as parameters, we use data members */
/*      even though this function could be more general - which is not useful */
```

```
/*      as far as VA is not concerned.                                          */
/*****************************************************************************/

    if (ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage - Top View")
    {
        MergedRectangle.Left     = min(StageObject->PrevPos_TOP.Left,
StageObject->NewPos_TOP.Left);
        MergedRectangle.Top      = min(StageObject->PrevPos_TOP.Top,
StageObject->NewPos_TOP.Top);
        MergedRectangle.Right    = max(StageObject->PrevPos_TOP.Right,
StageObject->NewPos_TOP.Right);
        MergedRectangle.Bottom   = max(StageObject->PrevPos_TOP.Bottom,
StageObject->NewPos_TOP.Bottom);
    }
}
//---------------------------------------------------------------------------

bool __fastcall TMainForm::IntersectRects(TRect ComparedRect, TRect
ModelOfComparison)
{
/*****************************************************************************
******************/
/* This member function (method) takes two rects given as parameters and tells if
they intersect    */
/*      one another.  When this method is called from inside a loop, a different
[ComparedRect]     */
/*      is taken and the method checks if it is inside the TRect that serves as
the model of        */
/*      the comparison (sort of a reference, in other words) as it is the same
throughout the       */
/*      calling loop.
*/
/*****************************************************************************
******************/

    if ( ComparedRect.Right  < ModelOfComparison.Left   ||
         ComparedRect.Left   > ModelOfComparison.Right  ||
         ComparedRect.Top    > ModelOfComparison.Bottom ||
         ComparedRect.Bottom < ModelOfComparison.Top      )
    {
        //The two rectangles DO NOT share some pixels - ComparedRect is OUTSIDE
ModelOfComparison
        return false;
    }
    else
    {
        //The two rectangles share some pixels - ComparedRect is INSIDE
ModelOfComparison
        return true;
    };
}
//---------------------------------------------------------------------------

TRect __fastcall TMainForm::GetObjectNewCoord_FRONT(int X, int Y, int XOffset, int
YOffset)
{
/*****************************************************************************
********************/
/* This function first calculates a possible position of the ClipRect on the
screen, although it      */
/*      may be not valid.  Then, that 2D-location is converted into a VA's 3D-
space location before   */
/*      [CheckBounds()] makes sure the StageObject is within VA's 3D-space bounds.
Finally,          */
```

```
/*      the 3D-location is converted back to a 2D-location.
*/
/* The result returned is the new (valid) location of the StageObject being moved
in FRONT view.      */
/*************************************************************************
*********************/

//STEP 1 : Calculate a possible location on screen (not necessarily valid) of the
object's projection

    TRect DestClipRect;

    //Records the possible location using the mouse pointer location and the
distance from it to the left and top sides
    DestClipRect.Left   = (X - XOffset);
    DestClipRect.Top    = (Y - YOffset);
    DestClipRect.Right  = (X - XOffset) + StageObject->CurWidth;
    DestClipRect.Bottom = (Y - YOffset) + StageObject->CurHeight;

//STEP 2 : Convert the 2D-location (on screen) into a 3D-location (in VA's 3D-
space)

    int MiddleLine;     //Vertical middle of the view
    TRect Location_3D;  //TRect holding the location of each sides in VA's 3D-
space

    //Converts the left side of the ClipRect first, the right side then
    MiddleLine = ImageStage->Width / 2;
    Location_3D.Left  = ConvertLocation_2Dto3D( MiddleLine, DestClipRect.Left ,
StageObject->Center_3D.z);
    Location_3D.Right = ConvertLocation_2Dto3D( MiddleLine, DestClipRect.Right,
StageObject->Center_3D.z);

    //Bottom side of the ClipRect with perspective first, top side then
    //You've got to look at VA's 3D-space from the right side and make it rotate
so that the top is on the left of the screen and the bottom on the right.
    MiddleLine = ImageStage->Height / 2;
    Location_3D.Top    = ConvertLocation_2Dto3D( MiddleLine, DestClipRect.Top   ,
StageObject->Center_3D.z);
    Location_3D.Bottom = ConvertLocation_2Dto3D( MiddleLine, DestClipRect.Bottom,
StageObject->Center_3D.z);

//STEP 3 : Make location in VA's 3D-space valid and work out [Center_3D]
    CheckBounds(Location_3D);

//STEP 4 : Convert the 3D-location (in VA's 3D-space) into a 2D-location (on
screen) and return result

    //To make the conversion, we just need a valid Center_3D in VA's 3D-space -
that task has been done during STEP 3.
    DestClipRect = DepthRendering();

    return DestClipRect;    //The location on scren is now valid
}
//---------------------------------------------------------------------------

TRect __fastcall TMainForm::GetObjectNewCoord_TOP(int X, int Y, int XOffset, int
YOffset)
{
/*************************************************************************
*************************/
/* This function first calculates a possible position of the ClipRect on the
screen, although it          */
```

```
/*      may be not valid.  Then, it makes sure that the 'polygon' is within the
bounds in TOP view.          */
/*      The result returned is the new (valid) location of the polygon being moved
in TOP view.              */
/* "Within the bounds" in TOP VIEW means that we can have 6 pixels drawn outside
of the bounds               */
/*    at the top and the bottom of the screen area representing the stage to allow
placing                   */
/*    the [Center_3D.z] exactly at the back of the stage (i.e. the top of the
screen area representing      */
/*    the stage, on 0) or at the front of the stage (i.e. the bottom of the screen
area representing         */
/*    the stage, on [ImageStage->Height-1] ).
*/
/* The X and Z coordinates of [Center_3D] are updated but not the Y because its
irrelevant in TOP VIEW      */
/*    For the X coord., we use the original size of the object because in TOP
VIEW, the polygons            */
/*    representing the object are not subjected to any perspective effect that
could shrink their            */
/*    ClipRect, their width.
*/
/*
*/
/* N.B. : @ "offset towards the top" means 'towards the top of the screen area
representing the stage'.     */
/*          It's an offset towards the 'back of the stage', actually.
*/
/*        @ [ ImageStage->Height-1 ] is used because the 522-nd line is at the
521-st position                */
/*        @ See comment at the top of this file to remember how [Center_3D.z] is
handled.                     */
/*        @ [Y - YOffset] is actually object's [Center_3D.z], that's why we use [Y
- YOffset] to set center */
/*****************************************************************************
**************************/

//STEP 1 : Calculate a possible location in TOP view (not necessarily valid) of
the object

    TRect DestClipRect;

    //Records the possible location using the mouse pointer location and the
distance from it to the left and top sides
    DestClipRect.Left   = (X - XOffset);
    DestClipRect.Top    = (Y - YOffset) - 6;
    DestClipRect.Right  = (X - XOffset) + StageObject->StorageBmp->Width;
    DestClipRect.Bottom = (Y - YOffset) + 6;

//STEP 2 : Make location valid and work out [Center_3D]

  //PART I : left and right sides first considered

    //Makes valid the left and right sides
    if (DestClipRect.Left < 0)
    {
        //Shifts left and right sides to a valid position - offset towards the
right
        DestClipRect.Left  = 0;
        DestClipRect.Right = 0 + StageObject->StorageBmp->Width;
    }
    else
    {
        if (DestClipRect.Right > (ImageStage->Width - 1))
```

```
        {
            //Shifts left and right sides to a valid position - offset towards the
left
            DestClipRect.Left  = (ImageStage->Width - 1) - StageObject-
>StorageBmp->Width;
            DestClipRect.Right =  ImageStage->Width - 1;
        }
        //ELSE : Left and right sides are within bounds and don't need to be
shifted
    }
    //Checks again the left side in the case the object could be larger than the
stage itself.
    //  If so, only the left side is modified - it's like shrinking the ClipRect.
    if (DestClipRect.Left < 0)
    {
        //Shrinks the ClipRect by displacing left side to a valid position -
offset towards the right
        DestClipRect.Left = 0;
    }

    //Works out [Center_3D] : only X is concerned.  Z is modified later on and Y
is left unchanged by [tsMove] in TOP
    //  If the size of the StageObject is larger than the stage, we can't use
[StageObject->StorageBmp->Width /2]
    int HalfWidth = (DestClipRect.Right - DestClipRect.Left) /2;
    StageObject->Center_3D.x = DestClipRect.Left + HalfWidth;

  //PART II : then, [Center_3D.z] is considered - and not top and bottom anymore -
to allow placing it on the top/bottom edge of the stage.
  //    Only Z is concerned.  X is modified here above and Y is left unchanged by
[tsMove] in TOP view

    //Makes valid the [Center_3D.z]
    if ((Y - YOffset) < 0)
    {
        //Shifts [Center_3D.z] to a valid position - offset towards the bottom
        StageObject->Center_3D.z = ImageStage->Height - 1;  //[Center_3D.z] is at
the back of the Stage
    }
    else
    {
        if ((Y - YOffset) > (ImageStage->Height - 1))
        {
            //Shifts top and bottom sides to a valid position - offset towards the
top
            StageObject->Center_3D.z = 0;                    //[Center_3D.z] is at
the front of the Stage
        }
        else //[Center_3D.z] is within bounds and don't need to be shifted
        {
            StageObject->Center_3D.z = (ImageStage->Height - 1) - (Y - YOffset);
        }
    }
    //Don't need to check again something because in TOP, the polygon's height
can't be taller than the stage itself.

    //Works out the top and bottom sides of [DestClipRect]
    DestClipRect.Top    = (ImageStage->Height - 1) - StageObject->Center_3D.z - 6;
    DestClipRect.Bottom = (ImageStage->Height - 1) - StageObject->Center_3D.z + 6;

    return DestClipRect;    //The location on scren is now valid
}
//---------------------------------------------------------------------------
```

```
TRect __fastcall TMainForm::DepthRendering()
{
/*****************************************************************************
**************************/
/* This function calls 4 times [ConvertLocation_3Dto2D].  The first time, we
consider a top view and        */
/*      we pass the coord. for the left and right sides of the object like if it
was seen from top.          */
/*      Then we pass the coord. of the bottom and top sides like if we were in a
side view - which           */
/*      doesn't actually exist. We also pass a [MiddleLine] that runs from the
back of the stage to         */
/*      the front in both views - although they are not placed on the same spot.
*/
/* Called 4 times, that triangle function gives us the TRect of the object's
projection on the screen's     */
/*      surface needed to get a perspective effect.  We just have to record the
current size of the         */
/*      object because it can be different from its original size if there's a
perspective effect.          */
/*
*/
/* N.B: The 3rd parameter of [ConvertLocation_3Dto2D()] is [StageObject-
>Center_3D.z] and not                */
/*      [(ImageStage->Height -1) - StageObject->Center_3D.z] because we consider a
distance (D1)             */
/*      that runs from the user's eye to the screen (600 units) and from the
screen to the object           */
/*      (given by [->Center_3D.z]) - so we don't need to take [ImageStage->Height
- 1] from which           */
/*      we substract [Center_3D.z] to get the displacement from the top of the
screen.                      */
/*****************************************************************************
**************************/

    int MiddleLine, HalfWidth;

    //It is the ClipRect of an object projected on the screen's surface (for
perspective effect) when this object is moved to the back of the screen
    TRect ClipRectProjectedOnScr;

//Works out the location on the screen's surface (for perspective effect) of the
ClipRect of an object which is moved to the back of the screen
    //Left side of the ClipRect with perspective first, right side then
    MiddleLine = ImageStage->Width / 2;
    HalfWidth  = StageObject->StorageBmp->Width  / 2;
    ClipRectProjectedOnScr.Left   = ConvertLocation_3Dto2D( MiddleLine,
StageObject->Center_3D.x - HalfWidth, StageObject->Center_3D.z);
    ClipRectProjectedOnScr.Right  = ConvertLocation_3Dto2D( MiddleLine,
StageObject->Center_3D.x + HalfWidth, StageObject->Center_3D.z);

    //Bottom side of the ClipRect with perspective first, top side then
    //You've got to look at VA's 3D-space from the right side and make it rotate
so that the top is on the left of the screen and the bottom on the right.
    MiddleLine = ImageStage->Height / 2;
    HalfWidth  = StageObject->StorageBmp->Height / 2;
    ClipRectProjectedOnScr.Top    = ConvertLocation_3Dto2D( MiddleLine,
StageObject->Center_3D.y - HalfWidth, StageObject->Center_3D.z);
    ClipRectProjectedOnScr.Bottom = ConvertLocation_3Dto2D( MiddleLine,
StageObject->Center_3D.y + HalfWidth, StageObject->Center_3D.z);

    //The object might currently have a size different from its original one
    StageObject->CurWidth  = (ClipRectProjectedOnScr.Right  -
ClipRectProjectedOnScr.Left);
```

```
    StageObject->CurHeight = (ClipRectProjectedOnScr.Bottom -
ClipRectProjectedOnScr.Top);

    //The results given here above are re-used to shrink or stretch the ClipRect -
NOT the bitmap itself which keeps its original size.
    return ClipRectProjectedOnScr;
}
//---------------------------------------------------------------------------

int __fastcall TMainForm::ConvertLocation_3Dto2D(int MiddleLine, int HorizCoord,
int VertCoord)
{
/*****************************************************************************
**************************/
/* When an object is moved backward or forward in VA's 3D space, because the
screen is in 2D we need to     */
/*      shrink or stretch it to give the illusion of depth.  Considering an object
not at the front         */
/*      of the stage, we have to determine its projection on the screen's surface
with an effect of         */
/*      perspective.  For that, we take one side of the object at a time and give
its location              */
/*      horizontally and vertically(^).  To work out the location on the screen,
we use two triangles       */
/*      with a rectangular corner.  The base of the biggest starts on user's
location and ends on the       */
/*      object - named [D1].  Its height is the distance between a middle line
(running from back-stage     */
/*      to front-stage) and the considered side of the object (either left or
right) - named [H1].          */
/*      The base of the smallest starts from the user again and ends on the screen
- named [D2].             */
/*      Its height runs from the middle again to its vertex on the screen's
surface after projection        */
/*      on it - named [H2]; it's the 'unknown'.
*/
/* Knowing that ( H1/D1 ) = ( H2/D2 ), we can calculate H2 and return the distance
from the left            */
/*      of the stage - that is the value for the middle line plus H2.
*/
/*
*/
/* N.B: (^) [VertCoord] starts at the bottom.  D1 runs from the user's eye to the
screen (600 units)        */
/*      and from the screen to the object (given by [VertCoord]) - so we don't
need to take                 */
/*      [ImageStage->Height - 1] from which we substract [Center_3D.z] to get the
displacement              */
/*      from the top of the screen.
*/
/*****************************************************************************
**************************/

    int D1 = UserToScreenDist + VertCoord;
    int D2 = UserToScreenDist;
    //If [HorizCoord] is smaller, we know that the point in VA's TOP VIEW is on
the left of the
    //      [MiddleLine] and so the result is negative - will be useful later on.
    float H1 = HorizCoord - MiddleLine;
    //Calculates the 'unknown', the horizontal location on the screen to give the
impression of depth
    float H2 = (H1 / D1) * D2;
```

```
    //If [H2] is negative, the location on the screen's surface is on the left of
the middle line.
    //      If not, it's on the right.  So we just need to add [H2] to the value
representing the middle line
    //      and we get the distance from the left of the stage - just what we need
to display.
    return (MiddleLine + H2);
}
//---------------------------------------------------------------------------

int __fastcall TMainForm::ConvertLocation_2Dto3D(int MiddleLine, int HorizCoord,
int VertCoord)
{
/*****************************************************************************
**************************/
/* It is the inverse of [ConvertLocation_3Dto2D].  We have the ClipRect for the
projection of the           */
/*      StageObject on screen and we need to know where it is placed in VA's 3D-
space.                      */
/* With this aim in view, we take one side of the ClipRect (on scr.) at a time and
give its location        */
/*      horizontally and vertically(^).  To work out the location in 3D-space, we
use two triangles          */
/*      with a rectangular corner.  The base of the biggest starts on user's
location and ends on the       */
/*      object - named [D2].  Its height is the distance between a middle line
(running from back-stage     */
/*      to front-stage) and the considered side of the object (either left or
right) - named [H2].          */
/*      The base of the smallest starts from the user again and ends on the screen
- named [D1].             */
/*      Its height runs from the middle again to its vertex on the screen's
surface after projection        */
/*      on it - named [H1].  [H2] is the 'unknown'.
*/
/* Knowing that ( H1/D1 ) = ( H2/D2 ), we can calculate H2 and return the distance
from the left             */
/*      of the stage - that is the value for the middle line plus H2.
*/
/*
*/
/* N.B: (^) [VertCoord] starts at the bottom.  D1 runs from the user's eye to the
screen (600 units)        */
/*      and from the screen to the object (given by [VertCoord]) - so we don't
need to take                */
/*      [ImageStage->Height - 1] from which we substract [Center_3D.z] to get the
displacement               */
/*      from the top of the screen.
*/
/*****************************************************************************
**************************/

    int D1 = UserToScreenDist;
    int D2 = UserToScreenDist + VertCoord;
    //If [HorizCoord] is smaller, we know that the point in VA's TOP VIEW is on
the left of the
    //      [MiddleLine] and so the result is negative - will be useful later on.
    float H1 = HorizCoord - MiddleLine;
    //Calculates the 'unknown', the horizontal location on the screen to give the
impression of depth
    float H2 = (H1 / D1) * D2;

    //If [H2] is negative, the location on the screen's surface is on the left of
the middle line.
```

```
    //      If not, it's on the right.  So we just need to add [H2] to the value
representing the middle line
    //      and we get the distance from the left of the stage - just what we need
to display.
    return (MiddleLine + H2);
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::CheckBounds(TRect ValidLocation_3D)
{
/*****************************************************************************
**************************/
/* This function makes sure the StageObject is within VA's 3D-space bounds.  With
this aim in view,         */
/*      it first considers the left and right sides in a top view and offset them
if they are not           */
/*      within the bounds.  It also takes into consideration the fact that an
object can be bigger          */
/*      than the stage itself.  Then, the same work is done for the top and bottom
sides but from           */
/*      a side view this time.
*/
/* For each view taken into consideration, only one coordinate of [Center_3D] is
calculated at              */
/*      a time.  Z is not modified by a normal move in X and Y directions.
*/
/*
*/
/* N.B. : @ "offset towards the top" means 'towards the top of the screen area
representing the stage'.     */
/*          In this case, it's an offset towards the 'actual top of the stage'.
*/
/*        @ [ImageStage->Height -1] is used because the 522-nd  line is at the
521-st  position             */
/*        @ [ImageStage->Width  -1] is used because the 1024-nd line is at the
1023-st position             */
/*****************************************************************************
**************************/

    int HalfWidth;

//PART I : left and right sides first considered

    //Considers a view from the 'TOP' to make valid the left and right sides
    if (ValidLocation_3D.Left < 0)
    {
        //Shifts left and right sides to a valid position - offset towards the
right
        ValidLocation_3D.Left  = 0;
        ValidLocation_3D.Right = 0 + StageObject->StorageBmp->Width;
    }
    else
    {
        if (ValidLocation_3D.Right > (ImageStage->Width - 1))
        {
            //Shifts left and right sides to a valid position - offset towards the
left
            ValidLocation_3D.Left  = (ImageStage->Width - 1) - StageObject-
>StorageBmp->Width;
            ValidLocation_3D.Right =  ImageStage->Width - 1;
        }
        //ELSE : Left and right sides are within bounds and don't need to be
shifted
    }
```

```
    //Checks again the left side in the case the object could be larger than the
stage itself.
    //  If so, only the left side is modified - it's like shrinking the ClipRect.
    if (ValidLocation_3D.Left < 0)
        //Shrinks the ClipRect by displacing left side to a valid position -
offset towards the right
        ValidLocation_3D.Left = 0;

    //Works out [Center_3D] : only X is concerned.  Y is modified later on and Z
is left unchanged by [tsMove]
    //  If the size of the StageObject is larger than the stage, we can't use
[StageObject->StorageBmp->Width /2]
    HalfWidth = (ValidLocation_3D.Right - ValidLocation_3D.Left) /2;
    StageObject->Center_3D.x = ValidLocation_3D.Left + HalfWidth;

//PART II : then, top and bottom considered

    //Considers a view from the 'SIDE' to make valid the top and bottom sides
    if (ValidLocation_3D.Top < 0)
    {
        //Shifts top and bottom sides to a valid position - offset towards the
bottom
        ValidLocation_3D.Top    = 0;
        ValidLocation_3D.Bottom = 0 + StageObject->StorageBmp->Height;
    }
    else
    {
        if (ValidLocation_3D.Bottom > (ImageStage->Height - 1))
        {
            //Shifts top and bottom sides to a valid position - offset towards the
top
            ValidLocation_3D.Top    = (ImageStage->Height - 1) - StageObject-
>StorageBmp->Height;
            ValidLocation_3D.Bottom =  ImageStage->Height - 1;
        }
        //ELSE : Top and bottom sides are within bounds and don't need to be
shifted
    }
    //Checks again the top side in the case the object could be taller than the
stage itself.
    //  If so, only the top side is modified - it's like shrinking the ClipRect.
    if (ValidLocation_3D.Top < 0)
        //Shrinks the ClipRect by displacing top side to a valid position - offset
towards the bottom
        ValidLocation_3D.Top  = 0;

    //Works out [Center_3D] : only Y is concerned.  X is modified here above and Z
is left unchanged by [tsMove]
    //  If the size of the StageObject is taller than the stage, we can't use
[StageObject->StorageBmp->Height /2]
    HalfWidth = (ValidLocation_3D.Bottom - ValidLocation_3D.Top) /2;
    StageObject->Center_3D.y = ValidLocation_3D.Top + HalfWidth;
}
//---------------------------------------------------------------------------
```

Tools.h (header file)

```
//---------------------------------------------------------------------------

#ifndef ToolsH
```

```
#define ToolsH

//---------------------------------------------------------------------------

#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <ComCtrls.hpp>
#include <ExtCtrls.hpp>
#include <Dialogs.hpp>
//---------------------------------------------------------------------------
class TToolBox : public TForm
{
__published:       // IDE-managed Components
    TStatusBar *ToolBoxStatusBar;
    TPanel *ToolBtnPickColour;
    TPanel *ToolBtnPickBlack;
    TPanel *ToolBtnPickWhite;
    TSpeedButton *ToolBtnNewWorld;
    TSpeedButton *ToolBtnOpenWorld;
    TSpeedButton *ToolBtnFrontView;
    TSpeedButton *ToolBtnInsertObject;
    TSpeedButton *ToolBtnStageClear;
    TSpeedButton *ToolBtnStageNew;
    TSpeedButton *ToolBtnTopView;
    TSpeedButton *ToolBtnBringToFront;
    TSpeedButton *ToolBtnBringForward;
    TSpeedButton *ToolBtnSaveWorld;
    TSpeedButton *ToolBtnSaveWorldAs;
    TSpeedButton *ToolBtnGoToStore;
    TSpeedButton *ToolBtnGoToWorkRoom;
    TSpeedButton *ToolBtnStageCopyAll;
    TSpeedButton *ToolBtnStagePasteAll;
    TSpeedButton *ToolBtnFirst;
    TSpeedButton *ToolBtnLast;
    TSpeedButton *ToolBtnSendBackward;
    TSpeedButton *ToolBtnSendToBack;
    TSpeedButton *ToolBtnRectangle;
    TSpeedButton *ToolBtnQuit;
    TSpeedButton *ToolBtnPlay;
    TSpeedButton *ToolBtnObjectToWorkRoom;
    TSpeedButton *ToolBtnShowWireframe;
    TSpeedButton *ToolBtnPrior;
    TSpeedButton *ToolBtnNext;
    TSpeedButton *ToolBtnStageDelete;
    TSpeedButton *ToolBtnStageInfo;
    TSpeedButton *ToolBtnClearWorkRoom;
    TSpeedButton *ToolBtnTakeWorkToStage;
    TSpeedButton *ToolBtnLine;
    TSpeedButton *ToolBtnUndo;
    TBevel *Bevel1;
    TBevel *Bevel2;
    TBevel *Bevel3;
    TSpeedButton *ToolBtnRedo;
    TSpeedButton *ToolBtnCircle;
    TSpeedButton *ToolBtnFreehandPencil;
    TSpeedButton *ToolBtnPenSize;
    TSpeedButton *ToolBtnEraser;
    TSpeedButton *ToolBtnCleanPixels;
    TSpeedButton *ToolBtnMove;
    TSpeedButton *ToolBtnMoveDeep;
    TSpeedButton *ToolBtnResizeObject;
```

```
TSpeedButton *ToolBtnFlipHorizontal;
TSpeedButton *ToolBtnFlipVertical;
TSpeedButton *ToolBtnResizePattern;
TSpeedButton *ToolBtnPattern;
TSpeedButton *ToolBtnTransparentObject;
TSpeedButton *ToolBtnPasteColour;
TSpeedButton *ToolBtnCutObject;
TSpeedButton *ToolBtnColourObject;
TSpeedButton *ToolBtnCopyObject;
TSpeedButton *ToolBtnDeleteObject;
TSpeedButton *ToolBtnPasteObject;
TSpeedButton *ToolBtnObjectToFloor;
TSpeedButton *ToolBtn3_8;
TSpeedButton *ToolBtn3_7;
void __fastcall FormCreate(TObject *Sender);
void __fastcall ToolBtnPickColourClick(TObject *Sender);
void __fastcall ToolBtnPickWhiteClick(TObject *Sender);
void __fastcall ToolBtnPickBlackClick(TObject *Sender);
void __fastcall ToolBtnNewWorldClick(TObject *Sender);
void __fastcall ToolBtnQuitClick(TObject *Sender);
void __fastcall ToolBtnOpenWorldClick(TObject *Sender);
void __fastcall ToolBtnSaveWorldAsClick(TObject *Sender);
void __fastcall ToolBtnPlayClick(TObject *Sender);
void __fastcall ToolBtnPriorClick(TObject *Sender);
void __fastcall ToolBtnNextClick(TObject *Sender);
void __fastcall ToolBtnStageClearClick(TObject *Sender);
void __fastcall ToolBtnInsertObjectClick(TObject *Sender);
void __fastcall ToolBtnStageNewClick(TObject *Sender);
void __fastcall ToolBtnSaveWorldClick(TObject *Sender);
void __fastcall ToolBtnFrontViewClick(TObject *Sender);
void __fastcall ToolBtnGoToWorkRoomClick(TObject *Sender);
void __fastcall ToolBtnObjectToWorkRoomClick(TObject *Sender);
void __fastcall ToolBtnTopViewClick(TObject *Sender);
void __fastcall ToolBtnGoToStoreClick(TObject *Sender);
void __fastcall ToolBtnShowWireframeClick(TObject *Sender);
void __fastcall ToolBtnStageCopyAllClick(TObject *Sender);
void __fastcall ToolBtnStageDeleteClick(TObject *Sender);
void __fastcall ToolBtnStagePasteAllClick(TObject *Sender);
void __fastcall ToolBtnStageInfoClick(TObject *Sender);
void __fastcall ToolBtnFirstClick(TObject *Sender);
void __fastcall ToolBtnLastClick(TObject *Sender);
void __fastcall ToolBtnBringToFrontClick(TObject *Sender);
void __fastcall ToolBtnSendToBackClick(TObject *Sender);
void __fastcall ToolBtnClearWorkRoomClick(TObject *Sender);
void __fastcall ToolBtnBringForwardClick(TObject *Sender);
void __fastcall ToolBtnSendBackwardClick(TObject *Sender);
void __fastcall ToolBtnTakeWorkToStageClick(TObject *Sender);
void __fastcall ToolBtnUndoClick(TObject *Sender);
void __fastcall ToolBtnRectangleClick(TObject *Sender);
void __fastcall ToolBtnLineClick(TObject *Sender);
void __fastcall ToolBtnRedoClick(TObject *Sender);
void __fastcall ToolBtnCircleClick(TObject *Sender);
void __fastcall ToolBtnFreehandPencilClick(TObject *Sender);
void __fastcall ToolBtnPenSizeClick(TObject *Sender);
void __fastcall ToolBtnEraserClick(TObject *Sender);
void __fastcall ToolBtnCleanPixelsClick(TObject *Sender);
void __fastcall ToolBtnMoveClick(TObject *Sender);
void __fastcall ToolBtnMoveDeepClick(TObject *Sender);
void __fastcall ToolBtnFlipHorizontalClick(TObject *Sender);
void __fastcall ToolBtnFlipVerticalClick(TObject *Sender);
void __fastcall ToolBtnResizeObjectClick(TObject *Sender);
void __fastcall ToolBtnPatternClick(TObject *Sender);
void __fastcall ToolBtnResizePatternClick(TObject *Sender);
void __fastcall ToolBtnTransparentObjectClick(TObject *Sender);
```

```
    void __fastcall ToolBtnPasteColourClick(TObject *Sender);
    void __fastcall ToolBtnColourObjectClick(TObject *Sender);
    void __fastcall ToolBtnCutObjectClick(TObject *Sender);
    void __fastcall ToolBtnCopyObjectClick(TObject *Sender);
    void __fastcall ToolBtnPasteObjectClick(TObject *Sender);
    void __fastcall ToolBtnDeleteObjectClick(TObject *Sender);
    void __fastcall ToolBtnObjectToFloorClick(TObject *Sender);
private:    // User declarations
public:            // User declarations
    __fastcall TToolBox(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TToolBox *ToolBox;
//---------------------------------------------------------------------------
#endif
```

Tools.cpp (file for handling the tool box)

```
//---------------------------------------------------------------------------

#include <vcl.h>

#pragma hdrstop



#include <stdlib.h>

#include "Tools.h"

#include "Main.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TToolBox *ToolBox;
//---------------------------------------------------------------------------
__fastcall TToolBox::TToolBox(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::FormCreate(TObject *Sender)
{
    ToolBox->Left = ((MainForm->ClientWidth - ToolBox->Width) / 2);
    ToolBox->Top = (MainForm->ClientHeight - ToolBox->Height) + 40;  //problems
with ClientHeight : menu's height not included
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnPickColourClick(TObject *Sender)
{
    MainForm->EditColourPickColourClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnPickWhiteClick(TObject *Sender)
{
    MainForm->EditColourPickWhiteClick(Sender);
}
```

```
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnPickBlackClick(TObject *Sender)
{
    MainForm->EditColourPickBlackClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnNewWorldClick(TObject *Sender)
{
    ToolBtnNewWorld->Down = false;
    MainForm->FileNewWorldClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnQuitClick(TObject *Sender)
{
    MainForm->FileQuitClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnOpenWorldClick(TObject *Sender)
{
    ToolBtnOpenWorld->Down = false;
    MainForm->FileOpenWorldClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnSaveWorldAsClick(TObject *Sender)
{
    ToolBtnSaveWorldAs->Down = false;
    MainForm->FileSaveWorldAsClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnPlayClick(TObject *Sender)
{
    ToolBtnPlay->Down = false;
    MainForm->FilePlayClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnPriorClick(TObject *Sender)
{
    ToolBtnPrior->Down = false;
    MainForm->StagePriorClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnNextClick(TObject *Sender)
{
    ToolBtnNext->Down = false;
    MainForm->StageNextClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnStageClearClick(TObject *Sender)
{
    ToolBtnStageClear->Down = false;
    MainForm->StageClearClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnInsertObjectClick(TObject *Sender)
```

```
{
    ToolBtnInsertObject->Down = false;
    MainForm->EditInsertObjectClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnStageNewClick(TObject *Sender)
{
    ToolBtnStageNew->Down = false;
    MainForm->StageNewClick(Sender);
}
//---------------------------------------------------------------------------


void __fastcall TToolBox::ToolBtnSaveWorldClick(TObject *Sender)
{
    ToolBtnSaveWorld->Down = false;
    MainForm->FileSaveWorldClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnFrontViewClick(TObject *Sender)
{
    ToolBtnFrontView->Down = false;
    MainForm->StageFrontViewClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnGoToWorkRoomClick(TObject *Sender)
{
    ToolBtnGoToWorkRoom->Down = false;
    MainForm->StageGoToWorkRoomClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnObjectToWorkRoomClick(TObject *Sender)
{
    ToolBtnObjectToWorkRoom->Down = false;
    MainForm->StageObjectToWorkRoomClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnTopViewClick(TObject *Sender)
{
    ToolBtnTopView->Down = false;
    MainForm->StageTopViewClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnGoToStoreClick(TObject *Sender)
{
    ToolBtnGoToStore->Down = false;
    MainForm->StageGoToStoreClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnShowWireframeClick(TObject *Sender)
{
    ToolBtnShowWireframe->Down = false;
    MainForm->StageShowWireframeClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnStageCopyAllClick(TObject *Sender)
```

```
{
    ToolBtnStageCopyAll->Down = false;
    MainForm->StageCopyAllClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnStageDeleteClick(TObject *Sender)
{
    ToolBtnStageDelete->Down = false;
    MainForm->StageDeleteClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnStagePasteAllClick(TObject *Sender)
{
    ToolBtnStagePasteAll->Down = false;
    MainForm->StagePasteAllClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnStageInfoClick(TObject *Sender)
{
    ToolBtnStageInfo->Down = false;
    MainForm->StageInfoClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnFirstClick(TObject *Sender)
{
    ToolBtnFirst->Down = false;
    MainForm->StageFirstClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnLastClick(TObject *Sender)
{
    ToolBtnLast->Down = false;
    MainForm->StageLastClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnBringToFrontClick(TObject *Sender)
{
    MainForm->WorkBringToFrontClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnSendToBackClick(TObject *Sender)
{
    MainForm->WorkSendToBackClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnClearWorkRoomClick(TObject *Sender)
{
    ToolBtnClearWorkRoom->Down = false;
    MainForm->WorkClearWorkRoomClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnBringForwardClick(TObject *Sender)
{
    MainForm->WorkBringForwardClick(Sender);
}
```

```
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnSendBackwardClick(TObject *Sender)
{
    MainForm->WorkSendBackwardClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnTakeWorkToStageClick(TObject *Sender)
{
    ToolBtnTakeWorkToStage->Down = false;
    MainForm->WorkTakeWorkToStageClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnUndoClick(TObject *Sender)
{
    ToolBtnUndo->Down = false;
    MainForm->EditUndoClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnRectangleClick(TObject *Sender)
{
    MainForm->EditShapeRectangleClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnLineClick(TObject *Sender)
{
    MainForm->EditShapeLineClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnRedoClick(TObject *Sender)
{
    ToolBtnRedo->Down = false;
    MainForm->EditRedoClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnCircleClick(TObject *Sender)
{
    MainForm->EditShapeCircleClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnFreehandPencilClick(TObject *Sender)
{
    MainForm->EditShapeFreeHandPencilClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnPenSizeClick(TObject *Sender)
{
    ToolBtnPenSize->Down = false;
    MainForm->EditPenSizeClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnEraserClick(TObject *Sender)
{
    MainForm->EditEraserClick(Sender);
}
```

```
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnCleanPixelsClick(TObject *Sender)
{
    MainForm->EditCleanPixelsClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnMoveClick(TObject *Sender)
{
    MainForm->EditMoveClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnMoveDeepClick(TObject *Sender)
{
    MainForm->EditMoveDeepClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnFlipHorizontalClick(TObject *Sender)
{
    MainForm->EditFlipHorizontalClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnFlipVerticalClick(TObject *Sender)
{
    MainForm->EditFlipVerticalClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnResizeObjectClick(TObject *Sender)
{
    MainForm->EditResizeObjectClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnPatternClick(TObject *Sender)
{
    MainForm->EditPatternClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnResizePatternClick(TObject *Sender)
{
    MainForm->EditResizePatternClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnTransparentObjectClick(TObject *Sender)
{
    MainForm->EditTransparentObjectClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnPasteColourClick(TObject *Sender)
{
    MainForm->EditColourPasteColourClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnColourObjectClick(TObject *Sender)
{
```

```
    MainForm->EditColourColourObjectClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnCutObjectClick(TObject *Sender)
{
    MainForm->EditCutObjectClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnCopyObjectClick(TObject *Sender)
{
    MainForm->EditCopyObjectClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnPasteObjectClick(TObject *Sender)
{
    ToolBtnPasteObject->Down = false;
    MainForm->EditPasteObjectClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnDeleteObjectClick(TObject *Sender)
{
    MainForm->EditDeleteObjectClick(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TToolBox::ToolBtnObjectToFloorClick(TObject *Sender)
{
    MainForm->EditObjectToFloorClick(Sender);
}
//---------------------------------------------------------------------------
```

About.h (header file)

```
//---------------------------------------------------------------------------

#ifndef AboutH

#define AboutH

//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TAboutBox : public TForm
{
__published:       // IDE-managed Components
    TImage *Image1;
    TBitBtn *AboutOk;
    TBevel *Bevel1;
    TLabel *Label3;
    TLabel *Label2;
    TLabel *Label1;
```

```
private:    // User declarations
public:           // User declarations
    __fastcall TAboutBox(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TAboutBox *AboutBox;
//---------------------------------------------------------------------------
#endif
```

About.cpp (About box)

```
//---------------------------------------------------------------------------

#include <vcl.h>

#pragma hdrstop


#include "About.h"

//---------------------------------------------------------------------------

#pragma package(smart_init)
#pragma resource "*.dfm"
TAboutBox *AboutBox;
//---------------------------------------------------------------------------
__fastcall TAboutBox::TAboutBox(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
```

Infos.h (header file)

```
//---------------------------------------------------------------------------
#ifndef InfosH
#define InfosH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------------
class TInfosBox : public TForm
{
__published:      // IDE-managed Components
    TLabel *Label1;
    TLabel *Label2;
    TLabel *ClWLab;
    TLabel *ClHLab;
    TLabel *Label3;
    TLabel *Label4;
    TLabel *WLab;
    TLabel *HLab;
    TLabel *Label5;
    TLabel *Label6;
    TLabel *WStage;
```

```
    TLabel *HStage;
private:    // User declarations
public:           // User declarations
    __fastcall TInfosBox(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TInfosBox *InfosBox;
//---------------------------------------------------------------------------
#endif
```

Infos.cpp (metrics about some objects)

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Infos.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TInfosBox *InfosBox;
//---------------------------------------------------------------------------
__fastcall TInfosBox::TInfosBox(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
```

PenSizing.h (header file)

```
//---------------------------------------------------------------------------
#ifndef PenSizingH
#define PenSizingH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include "cspin.h"
#include <Buttons.hpp>
#include <ComCtrls.hpp>
//---------------------------------------------------------------------------
class TPenSizeForm : public TForm
{
__published:      // IDE-managed Components
    TBitBtn *OkBtn;
    TBitBtn *CancelBtn;
    TGroupBox *GroupBox;
    TUpDown *UpDown;
    TEdit *EditSize;
private:    // User declarations
public:           // User declarations
    __fastcall TPenSizeForm(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TPenSizeForm *PenSizeForm;
```

```
//---------------------------------------------------------------------------
#endif
```

PenSizing.cpp (tool for changing the pen size)

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "PenSizing.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "cspin"
#pragma resource "*.dfm"
TPenSizeForm *PenSizeForm;
//---------------------------------------------------------------------------
__fastcall TPenSizeForm::TPenSizeForm(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
```

Code of the PC Prototype of Visual Assistant (Shortened version)

Vaprot.cpp (root file – execute code in *Main.cpp*)

```
//---------------------------------------------------------------------------

#include <vcl.h>

#pragma hdrstop

USERES("VAprot.res");

USEFORM("Main.cpp", MainForm);

USEFORM("Tools.cpp", ToolBox);

USEFORM("About.cpp", AboutBox);
USEFORM("Infos.cpp", InfosBox);
USEFORM("PenSizing.cpp", PenSizeForm);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    try
    {
        Application->Initialize();
        Application->Title = "Visual Assistant Prototype";
        Application->CreateForm(__classid(TMainForm), &MainForm);
        Application->CreateForm(__classid(TToolBox), &ToolBox);
        Application->CreateForm(__classid(TAboutBox), &AboutBox);
        Application->CreateForm(__classid(TInfosBox), &InfosBox);
        Application->CreateForm(__classid(TPenSizeForm), &PenSizeForm);
        Application->Run();
    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
    return 0;
}
//---------------------------------------------------------------------------
```

Main.h (Header file for *Main.cpp*)

```
//---------------------------------------------------------------------------
#ifndef MainH
#define MainH
//---------------------------------------------------------------------------
#include <vcl\sysutils.hpp>
#include <vcl\windows.hpp>
#include <vcl\messages.hpp>
#include <vcl\sysutils.hpp>
#include <vcl\classes.hpp>
#include <vcl\graphics.hpp>
```

```
#include <vcl\controls.hpp>
#include <vcl\forms.hpp>
#include <vcl\dialogs.hpp>
#include <vcl\stdctrls.hpp>
#include <vcl\buttons.hpp>
#include <vcl\extctrls.hpp>
#include <vcl\menus.hpp>
#include <Classes.hpp>
#include <Dialogs.hpp>
#include <Menus.hpp>
#include <Controls.hpp>
#include <ExtCtrls.hpp>
#include <ExtDlgs.hpp>
//---------------------------------------------------------------------------
enum TVATools {tsNoTool,
               tsNewWorld, tsOpenWorld, tsSave, tsSaveAs,
               tsTopView,
               tsInsert, tsRectangle, tsCircle, tsLine, tsFreehand, tsEraser,
tsPensize,
               tsMove, tsTransp,
               tsDelObj};
enum TWireframeMode {wfShow, wfHide};
enum TDisplayMode   {dmWhole, dmObjectClipRect_FrontView,
dmObjectClipRect_TopView, dmMergedClipRect};


class TMainForm : public TForm
{
__published:

      TMainMenu *MainMenu;
    TMenuItem *FileNewWorld;
    TMenuItem *FileOpenWorld;
    TMenuItem *FileSaveWorldAs;
    TMenuItem *FileQuit;
    TMenuItem *EditUndo;
    TMenuItem *EditCutObject;
    TMenuItem *EditCopyObject;
    TMenuItem *EditPasteObject;
    TMenuItem *HelpAbout;
    TMenuItem *Sep2;
    TMenuItem *FilePlay;
    TMenuItem *EditRedo;
    TMenuItem *Sep11;
    TMenuItem *EditMove;
    TMenuItem *EditMoveDeep;
    TMenuItem *Sep10;
    TMenuItem *EditDeleteObject;
    TMenuItem *StageMenu;
    TMenuItem *StageFrontView;
    TMenuItem *StageTopView;
    TMenuItem *StageGoToStore;
    TMenuItem *StageGoToWorkRoom;
    TMenuItem *Sep4;
    TMenuItem *StageCopyAll;
    TMenuItem *StageDelete;
    TMenuItem *Sep5;
    TMenuItem *StagePrior;
    TMenuItem *Sep3;
    TMenuItem *StageShowWireframe;
    TMenuItem *HelpVA;
    TMenuItem *StagePasteAll;
    TMenuItem *StageClear;
    TMenuItem *StageNew;
```

```
TOpenDialog *OpenDialog;
TSaveDialog *SaveDialog;
TOpenPictureDialog *OpenPictureDialog;
TMenuItem *StageLast;
TMenuItem *StageFirst;
TMenuItem *FileSaveWorld;
TMenuItem *StageObjectToWorkRoom;
TMenuItem *StageInfo;
TMenuItem *Sep9;
TMenuItem *EditEraser;
TMenuItem *EditResizeObject;
TMenuItem *EditFlipHorizontal;
TMenuItem *EditFlipVertical;
TMenuItem *EditPattern;
TMenuItem *EditResizePattern;
TMenuItem *EditTransparentObject;
TMenuItem *EditColour;
TMenuItem *EditColourPickColour;
TMenuItem *EditColourPasteColour;
TMenuItem *EditColourColourObject;
TMenuItem *EditColourPickWhite;
TMenuItem *EditColourPickBlack;
TMenuItem *EditObjectToFloor;
TMenuItem *WorkMenu;
TMenuItem *EditInsertObject;
TMenuItem *EditDrawShapes;
TMenuItem *EditShapeRectangle;
TMenuItem *EditShapeCircle;
TMenuItem *EditShapeLine;
TMenuItem *EditShapeFreeHandPencil;
TMenuItem *EditPenSize;
TMenuItem *WorkGoToStageFront;
TMenuItem *WorkGoToStageTop;
TMenuItem *Sep6;
TMenuItem *WorkBringToFront;
TMenuItem *WorkBringForward;
TMenuItem *WorkSendBackward;
TMenuItem *WorkSendToBack;
TMenuItem *Sep7;
TMenuItem *WorkClearWorkRoom;
TMenuItem *WorkTakeWorkToStage;
TMenuItem *StageNext;
TMenuItem *Sep1;
TMenuItem *Sep8;
TMenuItem *Sep12;
TColorDialog *ColorDialog;
TImage *ImageStage;
TBevel *Bevel1;
TMenuItem *EditCleanPixels;

void __fastcall FileQuitClick(TObject *Sender);
void __fastcall HelpAboutClick(TObject *Sender);
void __fastcall StageShowWireframeClick(TObject *Sender);
void __fastcall FileNewWorldClick(TObject *Sender);
void __fastcall FileOpenWorldClick(TObject *Sender);
void __fastcall FileSaveWorldAsClick(TObject *Sender);
void __fastcall EditUndoClick(TObject *Sender);
void __fastcall EditInsertObjectClick(TObject *Sender);
void __fastcall StageNewClick(TObject *Sender);
void __fastcall EditTransparentObjectClick(TObject *Sender);
void __fastcall EditColourPickWhiteClick(TObject *Sender);
void __fastcall EditColourPickBlackClick(TObject *Sender);
void __fastcall EditColourPickColourClick(TObject *Sender);
void __fastcall FileSaveWorldClick(TObject *Sender);
```

```
    void __fastcall FilePlayClick(TObject *Sender);
    void __fastcall StageFrontViewClick(TObject *Sender);
    void __fastcall StageTopViewClick(TObject *Sender);
    void __fastcall StageGoToStoreClick(TObject *Sender);
    void __fastcall StageGoToWorkRoomClick(TObject *Sender);
    void __fastcall StageObjectToWorkRoomClick(TObject *Sender);
    void __fastcall StageClearClick(TObject *Sender);
    void __fastcall StageCopyAllClick(TObject *Sender);
    void __fastcall StagePasteAllClick(TObject *Sender);
    void __fastcall StageDeleteClick(TObject *Sender);
    void __fastcall StageInfoClick(TObject *Sender);
    void __fastcall StagePriorClick(TObject *Sender);
    void __fastcall StageNextClick(TObject *Sender);
    void __fastcall StageFirstClick(TObject *Sender);
    void __fastcall StageLastClick(TObject *Sender);
    void __fastcall WorkGoToStageFrontClick(TObject *Sender);
    void __fastcall WorkGoToStageTopClick(TObject *Sender);
    void __fastcall WorkBringToFrontClick(TObject *Sender);
    void __fastcall WorkBringForwardClick(TObject *Sender);
    void __fastcall WorkSendBackwardClick(TObject *Sender);
    void __fastcall WorkSendToBackClick(TObject *Sender);
    void __fastcall WorkClearWorkRoomClick(TObject *Sender);
    void __fastcall WorkTakeWorkToStageClick(TObject *Sender);
    void __fastcall EditRedoClick(TObject *Sender);
    void __fastcall EditShapeRectangleClick(TObject *Sender);
    void __fastcall EditShapeCircleClick(TObject *Sender);
    void __fastcall EditShapeLineClick(TObject *Sender);
    void __fastcall EditShapeFreeHandPencilClick(TObject *Sender);
    void __fastcall EditPenSizeClick(TObject *Sender);
    void __fastcall EditEraserClick(TObject *Sender);
    void __fastcall EditCleanPixelsClick(TObject *Sender);
    void __fastcall EditMoveClick(TObject *Sender);
    void __fastcall EditMoveDeepClick(TObject *Sender);
    void __fastcall EditResizeObjectClick(TObject *Sender);
    void __fastcall EditFlipHorizontalClick(TObject *Sender);
    void __fastcall EditFlipVerticalClick(TObject *Sender);
    void __fastcall EditPatternClick(TObject *Sender);
    void __fastcall EditResizePatternClick(TObject *Sender);
    void __fastcall EditColourPasteColourClick(TObject *Sender);
    void __fastcall EditColourColourObjectClick(TObject *Sender);
    void __fastcall EditCutObjectClick(TObject *Sender);
    void __fastcall EditCopyObjectClick(TObject *Sender);
    void __fastcall EditPasteObjectClick(TObject *Sender);
    void __fastcall EditDeleteObjectClick(TObject *Sender);
    void __fastcall EditObjectToFloorClick(TObject *Sender);
    void __fastcall HelpVAClick(TObject *Sender);

/***** EVENT Functions ************************************************/
    void __fastcall ImageStageMouseDown(TObject *Sender,
          TMouseButton Button, TShiftState Shift, int X, int Y);
    void __fastcall ImageStageMouseMove(TObject *Sender, TShiftState Shift,
          int X, int Y);
    void __fastcall ImageStageMouseUp(TObject *Sender, TMouseButton Button,
          TShiftState Shift, int X, int Y);
    void __fastcall FormCreate(TObject *Sender);



private:        // private user declarations

/***** Functions for INITIALIZING *************************************/
    void __fastcall CreateWorld();  //A "World" is a list of Stages
    void __fastcall CreateStage();  //A "Stage" is a list of Objects
    void __fastcall DeleteWorld();
```

```
    void __fastcall DeleteStage();
/***** MISC. : Functions for SAVING, UPDATING, etc. ***********************/
    TModalResult __fastcall CheckSave();
    void __fastcall UpdateFunctionsAvailable();
    void __fastcall WrappingUp();
    void __fastcall SortStageObjects();
/***** Functions for SELECTING ********************************************/
    void __fastcall SelectObject_FRONT(int X, int Y);
    void __fastcall SelectObject_TOP(int X, int Y);
/***** Functions for REFRESHING the FRONT VIEW *****************************/
    void __fastcall Rebuild_FRONT(TDisplayMode DisplayMode);
    void __fastcall CreateBlankStage_OffScr();
    void __fastcall S_H_Wireframe_OffScr();
    void __fastcall PaintObjects_OffScr(TDisplayMode DisplayMode);
    void __fastcall Display_OnScr(TDisplayMode DisplayMode);
/***** Functions for REFRESHING the TOP VIEW ******************************/
    void __fastcall Rebuild_TOP(TDisplayMode DisplayMode);
    void __fastcall CreateBlankFloor_OffScr();
    void __fastcall DrawTopRects_OffScr(TDisplayMode DisplayMode);
    //void __fastcall Display_OnScr(TDisplayMode DisplayMode);  --> The same
method is used to display what's off-screen
/***** Functions for DRAWING SHAPES ***************************************/
    void __fastcall DrawTempShape(POINT MouseLocOnClick, POINT MouseLocOnRelease);
    void __fastcall StoreShape   (POINT MouseLocOnClick, POINT MouseLocOnRelease);
/***** Functions for MOVING ***********************************************/
    void __fastcall MergeRects_FRONT();
    void __fastcall MergeRects_TOP();
    bool __fastcall IntersectRects(TRect ComparedRect, TRect ModelOfComparison);
   TRect __fastcall GetObjectNewCoord_FRONT(int X, int Y, int XOffset, int
YOffset);
   TRect __fastcall GetObjectNewCoord_TOP(int X, int Y, int XOffset, int YOffset);
/***** Functions for VA's 3D-SPACE ****************************************/
   TRect __fastcall DepthRendering();
     int __fastcall ConvertLocation_3Dto2D(int MiddleLine, int HorizCoord, int
VertCoord);
     int __fastcall ConvertLocation_2Dto3D(int MiddleLine, int HorizCoord, int
VertCoord);
    void __fastcall CheckBounds(TRect ValidLocation_3D);


public:         // public user declarations
      virtual __fastcall TMainForm(TComponent* Owner);

/***** GENERAL information for WORLD, STAGES & OBJECTS
***************************/
    bool                WorldModified;                  //Has the world been
modified since the last saving ?
    TModalResult        OpCancelled;                    //Has the user cancelled
the last [QUIT / NEW / OPEN] operation ?
    TList*              ListOfObjects;                  //A stage is made up of
objects stored in a list.
    int                 LastObjectID;                   //Global variable to
generate ID numbers for StageObjects
    int                 ObjectToSelect;                 //Used to extract form
ListOfObjects the object on which the current tool will be applied
    TVATools            PrevTool, CurTool;              //Data members to hold
current and previous tools
    int                 UserToScreenDist;               //Measures the distance
between the user and the screen's surface
/***** FRONT VIEW information for OFF-SCREEN & STAGE-SET
**************************/
    Graphics::TBitmap*  Workbench;                      //Off-screen bitmap
    TColor              StageColor, FloorColor;         //Keeps trace of the color
currently used for the Stage and the floor
```

```
    TWireframeMode      WireframeMode;                  //Tells if the Wireframe
has to be shown or hidden
/***** FRONT VIEW information for DRAWING
*****************************************/
    bool                CanDraw, CanDrawFreehand;       //Can I draw ? Can I do it
freehanded ?
    POINT               Origin, MovePt;                 //Structure with X and Y
fields to draw by dragging the mouse
    TRect               FreeHClipRect;                  //Clipping rectangle used
to draw with pencil (freehand)
    int                 CurrentPenSize;                 //Gives the size to use
when setting the Pen's width
/***** FRONT VIEW information for MOVING
*****************************************/
    bool                CanMove;                        //Can I move the object
I've selected ?
    TRect               MergedRectangle;                //Rect that includes the 2
rects defined here above
    int                 XOffset, YOffset;               //Distance between the
MouseLoc.[X/Y] and the left/top side of the clip.rect.,
                                                        //  but in TOP VIEW,
YOffset is the distance between the MouseLoc.[Y] and the object's [Center_3D.z] -
and not the top side anymore
};
//---------------------------------------------------------------------------

class TStageObject
{
public:           // User declarations
    __fastcall TStageObject();        //Constructor

/***** USER TYPES ****************************************/
typedef struct {
    int x, y, z;
}Point_3D;
/***** GENERAL information *******************************/
    int                 ID;                             //Identification number
(of course)
    Graphics::TBitmap*  StorageBmp;                     //Bitmap storing an object
of the stage
    //To get the object's transparency,        read the [TBitmap->Transparent]
property
    //To get the object's ORIGINAL dimensions, read the [TBitmap->Width] and [ -
>Height] properties
/***** VA'S 3D SPACE information *************************/
    Point_3D            Center_3D;                      //Coordinates of the
center in 3D space
/***** FRONT VIEW information ****************************/
    TRect               PrevPos_FRONT, NewPos_FRONT;    //Position ON THE SCREEN
area representing the stage (and not in VA's 3D-space) of the clipping rectangles
needed to move (deep) the object
    int                 CurWidth, CurHeight;            //Object's CURRENT
dimensions, that is the (shrinked) dimensions of the object's projection on the
screen's surface
/***** TOP VIEW information ******************************/
    TRect               PrevPos_TOP, NewPos_TOP;        //Position ON THE SCREEN
area representing the stage (and not in VA's 3D-space) of the clipping rectangles
needed to move (deep) the object

};
//---------------------------------------------------------------------------
extern PACKAGE TMainForm *MainForm;
//---------------------------------------------------------------------------
#endif
```

Main.cpp (master file calling all the other modules)

```
//---------------------------------------------------------------------------

#include <vcl\vcl.h>

#include <stdlib.h>


#pragma hdrstop


#include "Main.h"

#include "Tools.h"

#include "About.h"
#include "Infos.h"
#include "PenSizing.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"

TMainForm *MainForm;
/*****************************************************************************
**************************/
/* Instantiates the class used to store one of the objects on stage.
*/
/* New memory will be allocated whenever it's necessary.
*/
/* Can't make it work if it's defined elsewhere like this : [TStageObject*
StageObject = new TStageObject;] */
/*      in the code, where its' needed.  So it's defined here - as sort of a
global variable.                */
/*****************************************************************************
**************************/
TStageObject* StageObject;

/***** LEGEND
*****************************************************************************
************/
/*              Symbol   |  Meaning
*/
/*-------------------------------------------------------------------------------
-------------------------*/
/*                  //   |  Line comment, permanent : do not remove.
*/
/*             //*****   |  Comment for guidance, to remove later on.
*/
/*            [] or ''   |  It's used to make it easier to spot data members' or
methods' names, or values   */
/*****************************************************************************
**************************/

/***** USAGE OF [Center_3D.z]
*****************************************************************************/
/*
*/
/* The point of origin for the axes in the 3D-SPACE is the Top-Left-Front corner.
When an object's         */
/*      [Center_3D.z] = 0, the object is at the front of the stage (which is a 3D-
space).  But in           */
```

```
/*      TOP VIEW, the Z-coordinate is starting at 0/the top of the screen area
representing the stage       */
/*      - which is the back of the stage in the 3D-space.
*/
/* That's why the value stored in [Center_3D.z] is '0' when the object is at the
front of the stage,        */
/*      that is the bottom of the screen area delimiting the stage (coord.
'ImageStage->Height -1').        */
/*      The value stored in [Center_3D.z] is 'ImageStage->Height -1' when the
object is at the back         */
/*      of the stage, that is the top of the screen area delimiting the stage
(coord. '0').                 */
/* To use [Center_3D.z] for setting coord on screen, we have to take [ImageStage-
>Height - 1] from          */
/*      which we substract [Center_3D.z] to get the displacement from the top of
the screen.                   */
/*
*/
/* N.B: Whatever the view is, no matter if there's a (perspective) depth effect,
the center we are          */
/*      considering is always the center in VA's 3D-space - and not the center of
the projection             */
/*      of the object in one of the views !
*/
/*****************************************************************************
**************************/

//---------------------------------------------------------------------------
__fastcall TMainForm::TMainForm(TComponent* Owner)
      : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::FormCreate(TObject *Sender)
{
//PART I : Creates the World
    //Calls the method that creates a new list of Stages - that is, a new World
    CreateWorld();

//PART II : Displays the World
    //WireframeMode is initialized here to show the wireframe when VA starts
    WireframeMode = wfShow;
    //Calls the method / member function that draws a blank stage + a wireframe
    Rebuild_FRONT(dmWhole);
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::CreateWorld()
/*****************************************************************************/
/* A "World" is a list of Stages                                             */
/* In this method, everything concerning one world is initialized here.      */
/*      That world is made up of only one stage and will be expanded later   */
/*****************************************************************************/
{
    //Sets the size of ImageStage
    ImageStage->Width  =  MainForm->ClientWidth;
    ImageStage->Height = (MainForm->ClientWidth * 0.51);  //Height is 51% of the
Width

    //Declares the bitmap representing the stage off-screen
    Workbench   = new Graphics::TBitmap;

    //Gives Workbench (offscreen bitmap) the size of a stage
```

```
    Workbench->Width  = ImageStage->Width;
    Workbench->Height = ImageStage->Height;

    //Sets a few other data members - e.g. CanMove, CurTool, ...
    WorldModified = false;                          //Sets Modified to false to
check later if work needs to be saved when user chooses NEW or OPEN
    LastObjectID = 0;                               //No objects at all on the
stage at this time
    ObjectToSelect = -1;                            //No object in ListOfObjects
has to undergo any operation at this time
    PrevTool = CurTool = tsNoTool;                  //No tool is selected at the
time of creation
    UserToScreenDist = 600;                         //The distance between the user
and the screen's surface is 600 units
    ColorDialog->Color = clTeal;                    //We need a default color for
drawing shapes
    StageColor = FloorColor = clNavy;               //clNavy is the default color
used for the Stage and the Floor
    CurrentPenSize = 3;                             //Default Pen Size for drawing
    CanDraw = CanDrawFreehand = CanMove = false;    //Sets all boolean to false -
safety measure
    Origin = MovePt = Point(0,0);                   //Starting coordinates
    XOffset = YOffset = 0;                          //No offset at the time of
creation

    //Calls the method that creates a new list of StageObjects - that is, a new
Stage
    CreateStage();

    //Updates the list of stages in Stage menu.

}
//---------------------------------------------------------------------------

void __fastcall TMainForm::CreateStage()
/*****************************************************************************/
/* A "Stage" is a list of Objects.                                           */
/* In this method, everything concerning one and only one stage that is,     */
/*      a list of objects, is initialized here.                              */
/*****************************************************************************/
{
    //A stage is made up of objects stored in a list.  At the time of creation,
only one empty list of objects is needed for the new world.
    ListOfObjects = new TList;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::DeleteWorld()
{
    //The off-screen bitmap has(?) to be deleted 'cos mem. has been dynamically
allocated to it - it doesn't exist at the design time.
    delete Workbench;

//***** Should loop through all the stages and delete them one by one by calling
//*****     [DeleteStage] with a argument indicating which Stage has to be deleted
    DeleteStage();
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::DeleteStage()
{
    //Frees the memory used by the objects listed in the TList of one stage.
    for (int i = 0; i < ListOfObjects->Count; i++)
    {
```

```
        StageObject = (TStageObject*) ListOfObjects->Items[i];
        delete StageObject;
    }
    //Then deletes the list itself.
    delete ListOfObjects;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::FileQuitClick(TObject *Sender)
{
    //Saves the current world if it's necessary.
    OpCancelled = CheckSave();

    if (OpCancelled != mrCancel)
    {
        //Calls the method that destroys all the Stages - that is, the World
        DeleteWorld();

        Application->Terminate();

    }//ELSE : do not quit VA because the user has cancelled the operation
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::FileNewWorldClick(TObject *Sender)
{
//***** Modify the LastStageObjectID

    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + FileNewWorld->Hint;
    PrevTool = CurTool;
    CurTool  = tsNewWorld;

    //Saves the current world if it's necessary.
    OpCancelled = CheckSave();

    if(OpCancelled != mrCancel)
    {
        //Disposes of the existing world.
        DeleteWorld();

        //Inits a new world made up of a blank stage only
        CreateWorld();
        //We need a default color for drawing shapes
        ToolBox->ToolBtnPickColour->Color = clTeal;     //It can't be done in
[createWorld] 'cos the ToolBox is not created first at the time of creation -
which results in an "ACCESS VIOLATION"

        //We don't need to save a blank world --> wait for further changes
        WorldModified = false;

        //Clears the screen and shows the Front View
        Rebuild_FRONT(dmWhole);
        ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text = " Stage - Front View";

        //Tells the user that a new world has been created and displayed
        ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " New World created
and displayed";

        //A few functionalities have to be (dis)activated in Front View
        UpdateFunctionsAvailable();

        //Empties the list of stages in Stage menu.
```

```
        //It's a new world, so we don't know its name yet
        SaveDialog->FileName = "";

    }//ELSE: do not clear the current World because the user has cancelled the
operation
    else
        //Tells the user that 'Create New World' has been cancelled
        ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " 'Create New World'
has been cancelled";
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::FileOpenWorldClick(TObject *Sender)
{
//***** Modify the LastStageObjectID

    //Modifies ToolBox appearance to reflect click on function  and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + FileOpenWorld->Hint;
    PrevTool = CurTool;
    CurTool  = tsOpenWorld;

    //Saves the current world if it's necessary.
    OpCancelled = CheckSave();

    if(OpCancelled != mrCancel)
    {
        //Disposes of the existing world.
        DeleteWorld();

        //Inits a new world made up of a blank stage only
        CreateWorld();
        //We need a default color for drawing shapes
        ToolBox->ToolBtnPickColour->Color = clTeal;     //It can't be done in
[createWorld] 'cos the ToolBox is not created first at the time of creation -
which results in an "ACCESS VIOLATION"

        //Expands the blank world created here above with information loaded from
a VA file
        OpenDialog->FileName = "";
        if (OpenDialog->Execute())
        {
            SaveDialog->FileName = OpenDialog->FileName;
            if (MessageDlg("Under construction - No loading at the moment ! \nOnly
a blank World is created.", mtWarning,
                    TMsgDlgButtons() << mbOK, 0) == mrOk)
                    //does nothing
                    ;
        }

        //We don't need to save a world thas has been loaded and, a fortiori,
previously saved --> wait for further changes
        WorldModified = false;

        //Clears the screen and displays in FRONT VIEW the World that has just
been loaded
        Rebuild_FRONT(dmWhole);
        ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text = " Stage - Front View";

        //Tells the user that a world has been loaded and displayed
        ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " A World has been
loaded and displayed";
```

```
        //A few functionalities have to be (dis)activated in Front View
        UpdateFunctionsAvailable();

        //Renews the list of stages in Stage menu.

    }//ELSE : do not clear the current World nor load a new World because the user
has cancelled the operation
    else
        //Tells the user that 'Open World' has been cancelled
        ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " 'Open World' has
been cancelled";
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::FileSaveWorldClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + FileSaveWorld->Hint;
    PrevTool = CurTool;
    CurTool  = tsSave;

    if (SaveDialog->FileName == "")
                FileSaveWorldAsClick(Sender);
    else
    {
        //SaveToFile();

        //Sets Modifed to false since we've just saved
        WorldModified = false;

        if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
              //does nothing
              ;

        //Tells the user that the world has been saved
        ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " The World has been
saved";
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::FileSaveWorldAsClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + FileSaveWorldAs-
>Hint;
    PrevTool = CurTool;
    CurTool  = tsSaveAs;

    if (SaveDialog->Execute())
    {
        //SaveToFile();

        //Sets Modifed to false since we've just saved
        WorldModified = false;

        if (MessageDlg("Under construction. \nJust believe that your world has
been saved !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
        //does nothing
```

```
        ;

        //Tells the user that the world has been saved with a new name
        ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " The World has been
saved as : '" + SaveDialog->FileName + "'";
    }
    else
        //Tells the user that the saving has been cancelled
        ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " Saving has been
cancelled";
}
//---------------------------------------------------------------------------

TModalResult __fastcall TMainForm::CheckSave()
{
    if (WorldModified)
    {
        switch(MessageDlg ("The current world has been changed. \nSave changes ?",
                           mtConfirmation, TMsgDlgButtons() <<mbYes <<mbNo
<<mbCancel, 0))
        {
            case mrYes      :
                //If YES was clicked, then save the world;
                FileSaveWorldAsClick(this);
                return mrYes;
            case mrNo       :
                //If NO was clicked, then don't save - just do nothing;
                return mrNo;
            case mrCancel   :
                //Tells the user that the saving has been cancelled
                ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " Saving has
been cancelled";

                //If CANCEL was clicked, then return - the return value is used in
[QUIT] to be able to cancel it.
                return mrCancel;
        }
    }
    //By default (and to avoid 'Compiler Warning'), the function returns [mrNone]
    return mrNone;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditUndoClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditUndo->Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditInsertObjectClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditInsertObject-
>Hint;
    PrevTool = CurTool;
    CurTool  = tsInsert;
```

```
    if (OpenPictureDialog->Execute())
    {
        //Loads the image on a temporary container (Type of it : TImage, in order
to use Autosize)
        TImage* TempImage = new TImage(this);
        TempImage->AutoSize = true;
        TempImage->Picture->LoadFromFile(OpenPictureDialog->FileName);

        //Allocates a new memory space for that object on stage and returns a
pointer to it
        StageObject = new TStageObject;
        //Gives an ID number to the object
        StageObject->ID = ++LastObjectID;

        //The bitmap's height and width properties hold the original size of the
object
        StageObject->StorageBmp->Width  = TempImage->Picture->Width;
        StageObject->StorageBmp->Height = TempImage->Picture->Height;

        //When an object is created, its current dimensions are identical to its
original dimensions
        StageObject->CurWidth  = StageObject->StorageBmp->Width;
        StageObject->CurHeight = StageObject->StorageBmp->Height;

        //Places and stores the position of the image on the stage
        StageObject->NewPos_FRONT.Left   = (ImageStage->Width  / 2) - (TempImage-
>Picture->Width  / 2);
        StageObject->NewPos_FRONT.Top    = (ImageStage->Height / 2) - (TempImage-
>Picture->Height / 2);
        StageObject->NewPos_FRONT.Right  = (ImageStage->Width  / 2) + (TempImage-
>Picture->Width  / 2);
        StageObject->NewPos_FRONT.Bottom = (ImageStage->Height / 2) + (TempImage-
>Picture->Height / 2);

        //Records the center in VA's 3D-space
        StageObject->Center_3D.x = ImageStage->Width  / 2;
        StageObject->Center_3D.y = ImageStage->Height / 2;
        StageObject->Center_3D.z = 0;

        //Copies Image from temp container to StorageBmp
        StageObject->StorageBmp->Assign(TempImage->Picture->Bitmap);
        delete TempImage;

        //Adds StageObject to the list of objects present on stage
        ListOfObjects->Add(StageObject);

        //Sets [ObjectToSelect] to make it reference to the image that has just
been inserted
        ObjectToSelect = (ListOfObjects->Count) - 1;       //It's the last index
because the list hasn't been sorted since

        //Display the image that has just been loaded off- and on screen
        Rebuild_FRONT(dmObjectClipRect_FrontView);

        //As a safety measure...
        WrappingUp();
    };
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageShowWireframeClick(TObject *Sender)
{
```

```
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageShowWireframe-
>Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    // Shows / Hides the wireframe
    if (StageShowWireframe->Checked)
    {
        //Unchecks the Wireframe item in the menu
        StageShowWireframe->Checked = false;

        WireframeMode = wfHide;
    }
    else
    {
        //Checks the Wireframe item in the menu
        StageShowWireframe->Checked = true;

        WireframeMode = wfShow;
    };
    Rebuild_FRONT(dmWhole);
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageNewClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageNew->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    //Adds a new stage to the world.

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditTransparentObjectClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBtnTransparentObject->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " +
EditTransparentObject->Hint;
    PrevTool = CurTool;
    CurTool  = tsTransp;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditColourPickWhiteClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditColourPickWhite-
>Hint;
    ToolBox->ToolBtnPickColour->Color = clWhite;
```

```
    ColorDialog->Color = clWhite;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditColourPickBlackClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditColourPickBlack-
>Hint;
    ToolBox->ToolBtnPickColour->Color = clBlack;
    ColorDialog->Color = clBlack;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditColourPickColourClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " +
EditColourPickColour->Hint;

    if (ColorDialog->Execute())
    {
        ToolBox->ToolBtnPickColour->Color = ColorDialog->Color;
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::FilePlayClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + FilePlay->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageFrontViewClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageFrontView-
>Hint;
    ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text = " Stage - Front View";
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    //Shows the Front View
    Rebuild_FRONT(dmWhole);

    //A few functionalities have to be (dis)activated in Front View
    UpdateFunctionsAvailable();

    //No function selected by default even though is the other way round in TOP
VIEW.
    ToolBox->ToolBtnMove->Down = false;
```

```
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " Please, choose a
function";
    PrevTool = CurTool;
    CurTool = tsNoTool;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageTopViewClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageTopView->Hint;
    ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text = " Stage - Top View";
    PrevTool = CurTool;
    CurTool  = tsTopView;

    //Shows the Top View
    Rebuild_TOP(dmWhole);

    //A few functionalities have to be (dis)activated in Top View
    UpdateFunctionsAvailable();

    //The function selected by default is [Move].
    ToolBox->ToolBtnMove->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditMove->Hint;
    PrevTool = CurTool;
    CurTool = tsMove;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageGoToStoreClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageGoToStore-
>Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
            TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageGoToWorkRoomClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageGoToWorkRoom-
>Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
            TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageObjectToWorkRoomClick(TObject *Sender)
```

```
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " +
StageObjectToWorkRoom->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageClearClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageClear->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageCopyAllClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageCopyAll->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StagePasteAllClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StagePasteAll->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageDeleteClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
```

```
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageDelete->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageInfoClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageInfo->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StagePriorClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StagePrior->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageNextClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageNext->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageFirstClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageFirst->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;
```

```
    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StageLastClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageLast->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::WorkGoToStageFrontClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageFrontView-
>Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    //Carry out a few things if necessary then call StageFrontView

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::WorkGoToStageTopClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + StageTopView->Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    //Carry out a few things if necessary then call StageTopView

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::WorkBringToFrontClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnBringToFront->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + WorkBringToFront-
>Hint;
```

```
    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::WorkBringForwardClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnBringForward->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + WorkBringForward-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::WorkSendBackwardClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnSendBackward->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + WorkSendBackward-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::WorkSendToBackClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnSendToBack->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + WorkSendToBack-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::WorkClearWorkRoomClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + WorkClearWorkRoom-
>Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------
```

```
void __fastcall TMainForm::WorkTakeWorkToStageClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + WorkTakeWorkToStage-
>Hint;
    PrevTool = CurTool;
    CurTool  = tsNoTool;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditRedoClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditRedo->Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditShapeRectangleClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnRectangle->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditShapeRectangle-
>Hint;
    PrevTool = CurTool;
    CurTool = tsRectangle;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditShapeCircleClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnCircle->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditShapeCircle-
>Hint;
    PrevTool = CurTool;
    CurTool = tsCircle;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditShapeLineClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnLine->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditShapeLine->Hint;
    PrevTool = CurTool;
    CurTool = tsLine;
```

```
    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditShapeFreeHandPencilClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnFreehandPencil->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " +
EditShapeFreeHandPencil->Hint;
    PrevTool = CurTool;
    CurTool = tsFreehand;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditPenSizeClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditPenSize->Hint;
    PrevTool = CurTool;
    CurTool = tsPensize;

    if (PenSizeForm->ShowModal() == mrOk)
    {
        CurrentPenSize = StrToInt(PenSizeForm->EditSize->Text);
    }
    else
    {   //keep the same Pen Size
        PenSizeForm->EditSize->Text = CurrentPenSize;
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditEraserClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnEraser->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditEraser->Hint;
    PrevTool = CurTool;
    CurTool = tsEraser;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditCleanPixelsClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnCleanPixels->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditCleanPixels-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
```

```
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditMoveClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnMove->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditMove->Hint;
    PrevTool = CurTool;
    CurTool = tsMove;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditMoveDeepClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnMoveDeep->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditMoveDeep->Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditResizeObjectClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnResizeObject->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditResizeObject-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditFlipHorizontalClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnFlipHorizontal->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditFlipHorizontal-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditFlipVerticalClick(TObject *Sender)
{
```

```
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnFlipVertical->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditFlipVertical-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditPatternClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnPattern->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditPattern->Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditResizePatternClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnResizePattern->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditResizePattern-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditColourPasteColourClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnPasteColour->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " +
EditColourPasteColour->Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditColourColourObjectClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
```

```
    ToolBox->ToolBtnColourObject->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " +
EditColourColourObject->Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditCutObjectClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnCutObject->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditCutObject->Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditCopyObjectClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnCopyObject->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditCopyObject-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditPasteObjectClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditPasteObject-
>Hint;
    ToolBox->ToolBtnPasteObject->Down = true;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditDeleteObjectClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function and disactivate
tool previously selected
    ToolBox->ToolBtnDeleteObject->Down = true;
```

```
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditDeleteObject-
>Hint;
    PrevTool = CurTool;
    CurTool  = tsDelObj;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::EditObjectToFloorClick(TObject *Sender)
{
    //Modifies ToolBox appearance to reflect click on function
    ToolBox->ToolBtnObjectToFloor->Down = true;
    ToolBox->ToolBoxStatusBar->Panels->Items[2]->Text = " " + EditObjectToFloor-
>Hint;

    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;

    //Wait for user to click on something ('MouseDown' event)
}
//---------------------------------------------------------------------------


void __fastcall TMainForm::HelpVAClick(TObject *Sender)
{
    if (MessageDlg("Under construction - Not implemented yet !", mtWarning,
              TMsgDlgButtons() << mbOK, 0) == mrOk)
    //does nothing
    ;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::HelpAboutClick(TObject *Sender)
{
    AboutBox->ShowModal();
//***** remove when not necessary anymore
    InfosBox->ClWLab->Caption = IntToStr(MainForm->ClientWidth);
    InfosBox->ClHLab->Caption = IntToStr(MainForm->ClientHeight);
    InfosBox->WLab->Caption   = IntToStr(MainForm->Width);
    InfosBox->HLab->Caption   = IntToStr(MainForm->Height);
    InfosBox->WStage->Caption = IntToStr(MainForm->ImageStage->Width);
    InfosBox->HStage->Caption = IntToStr(MainForm->ImageStage->Height);

    InfosBox->ShowModal();
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::SelectObject_FRONT(int X, int Y)
{
    //We assume that the object on stage is not selected by default
    ObjectToSelect = -1;

    for (int i = 0; i < ListOfObjects->Count; i++)
    {
        //Extracts the object at the i-th position in the list of objects present
on stage
        StageObject = (TStageObject*) ListOfObjects->Items[i];

        //Is the mouse location inside the clipping rectangle ?
```

```
        if ((X > StageObject->NewPos_FRONT.Left) && (Y > StageObject-
>NewPos_FRONT.Top) && (X < StageObject->NewPos_FRONT.Right) && (Y < StageObject-
>NewPos_FRONT.Bottom))
        {
            //Are the object's white pixels transparent ?
            if (!StageObject->StorageBmp->Transparent)
            {
                //No transparency
                ObjectToSelect = i;     //The object has been selected
            }
            else //Transparency
            {
                //Is the pixel under mouse loc white (white = TransparentColor) or
not ?
                if (StageObject->StorageBmp->Canvas->Pixels[X-StageObject-
>NewPos_FRONT.Left][Y-StageObject->NewPos_FRONT.Top] != clWhite)
                //In TrueColor-32bits, [StageObject->StorageBmp->TransparentColor]
ever holds 50331647 when clWhite = 16777215 ! So it's not possible to compare with
[TransparentColor]
                {
                    //Pixel under mouse loc is not white/transparent
                    ObjectToSelect = i;     //The object has been selected
                }
            }
        }
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::SelectObject_TOP(int X, int Y)
{
    //We assume that the object on stage is not selected by default
    ObjectToSelect = -1;

    for (int i = 0; i < ListOfObjects->Count; i++)
    {
        //Extracts the object at the i-th position in the list of objects present
on stage
        StageObject = (TStageObject*) ListOfObjects->Items[i];

        //Is the mouse location inside the clipping rectangle ?
        if ((X > StageObject->NewPos_TOP.Left) && (Y > StageObject-
>NewPos_TOP.Top) && (X < StageObject->NewPos_TOP.Right) && (Y < StageObject-
>NewPos_TOP.Bottom))
        {
            //Are the object's white pixels transparent ?
            if (!StageObject->StorageBmp->Transparent)
            {
                //No transparency
                ObjectToSelect = i;     //The object has been selected
            }
            else //Transparency
            {
                //Is the pixel under mouse loc white (white = TransparentColor) or
not ?
                if (StageObject->StorageBmp->Canvas->Pixels[X-StageObject-
>NewPos_FRONT.Left][Y-StageObject->NewPos_FRONT.Top] != clWhite)
                //In TrueColor-32bits, [StageObject->StorageBmp->TransparentColor]
ever holds 50331647 when clWhite = 16777215 ! So it's not possible to compare with
[TransparentColor]
                {
                    //Pixel under mouse loc is not white/transparent
                    ObjectToSelect = i;     //The object has been selected
                }
```

```
            }
        }
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::ImageStageMouseDown(TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)
{
    switch (CurTool)
    {
        case tsRectangle :
        case tsCircle :
        case tsLine :
            CanDraw = true;
            Origin  = Point(X,Y);
            MovePt  = Point(X,Y);
            break;
        case tsFreehand :
            CanDrawFreehand = true;

            //Sets pen and brush properties to paint Workbench's surface in white
first
            Workbench->Canvas->Pen->Style   = psSolid;
            Workbench->Canvas->Pen->Mode    = pmCopy;
            Workbench->Canvas->Pen->Color   = clWhite;
            Workbench->Canvas->Brush->Style = bsSolid;
            Workbench->Canvas->Brush->Color = clWhite;

            //Wipes the temporary bmp in white
            Workbench->Canvas->Rectangle(0, 0, Workbench->Width, Workbench-
>Height);

            //Sets pen with the color anf the pen size choosed by the user to draw
off-screen
            Workbench->Canvas->Pen->Color   = ColorDialog->Color;
            Workbench->Canvas->Pen->Width   = CurrentPenSize;

            //Sets pen to draw with on screen
            ImageStage->Canvas->Pen->Style = psSolid;
            ImageStage->Canvas->Pen->Mode  = pmCopy;
            ImageStage->Canvas->Pen->Color = ColorDialog->Color;

            //Places the pen to be ready to draw on both off- and onscreen
            Workbench->Canvas->MoveTo(X, Y);     //Places the pen off-screen on
Workbench
            ImageStage->Canvas->MoveTo(X, Y);    //Places the pen on screen to be
ready to give feedback to the user

            //Sizes a clip.rect. including the pixels colored (freehanded) by the
pencil
            FreeHClipRect = Rect(X, Y, X, Y);
            break;

        case tsMove :
            //We assume that the object on stage is not selected by default and so
can't be moved
            CanMove = false;

            //Calls the appropriate function that determines if an object has been
selected either in FRONT VIEW or in TOP VIEW
            if(ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage -
Front View")
            {//The current view is Front View
```

```
                SelectObject_FRONT(X, Y);

                if (ObjectToSelect > -1)
                {
                    //Extracts the object on which the current tool will be
applied
                    StageObject = (TStageObject*) ListOfObjects-
>Items[ObjectToSelect];

                    CanMove = true;

                    //Calculates the offset between (X,Y) MouseLoc and the left
and top side of the ClipRect
                    XOffset = X - StageObject->NewPos_FRONT.Left;
                    YOffset = Y - StageObject->NewPos_FRONT.Top;

                }//else : select nothing - the user has not clicked on an object
            }
            else //->Panels->Items[0]->Text == " Stage - Top View"
            {
                SelectObject_TOP(X, Y);

                if (ObjectToSelect > -1)
                {
                    //Extracts the object on which the current tool will be
applied
                    StageObject = (TStageObject*) ListOfObjects-
>Items[ObjectToSelect];

                    CanMove = true;

                    //Calculates the offset needed not to place ClipRect's TopLeft
corner in (X,Y) MouseLoc
                    XOffset = X - StageObject->NewPos_TOP.Left;
                    YOffset = Y - ((ImageStage->Height - 1) - StageObject-
>Center_3D.z);    //[YOffset] is the distance between MouseLoc's Y and Center_3D.z

                }//else : select nothing - the user has not clicked on an object
            }
            break;

        case tsTransp :
            //Calls the function that determines if an object has been selected
            SelectObject_FRONT(X, Y);

            if (ObjectToSelect > -1)
            {
                //Extracts the object on which the current tool will be applied
                StageObject = (TStageObject*) ListOfObjects-
>Items[ObjectToSelect];

                //Determines if the selected object must be drawn with
transparency or not
                StageObject->StorageBmp->TransparentColor = clWhite;     //Sets
the color used to apply the transparency effect
                StageObject->StorageBmp->Transparent = !StageObject->StorageBmp-
>Transparent;
                //Calls the method / member function that rebuilds the
StageObjects overlapping the clip.rect. of the modified object
                Rebuild_FRONT(dmObjectClipRect_FrontView);

                //As a safety measure...
                WrappingUp();
```

```
            }//else : select nothing - the user has not clicked on an object
            break;

        case tsDelObj :
            //Calls the appropriate function that determines if an object has been
selected
            if(ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage -
Front View")
            {//The current view is Front View
                SelectObject_FRONT(X, Y);
            }
            else //->Panels->Items[0]->Text == " Stage - Top View"
            {
                SelectObject_TOP(X, Y);
            }

            if (ObjectToSelect > -1)
            {
                //Extracts the object that will be deleted
                StageObject = (TStageObject*) ListOfObjects-
>Items[ObjectToSelect];

                //Frees the memory associated with the object stored at position
[ObjectToSelect]
                delete StageObject;
                StageObject = NULL;
                //Remove the entry in the TList
                ListOfObjects->Delete(ObjectToSelect);

                //The object currently selected has been deleted, so it's no use
to reference it anymore
                ObjectToSelect = -1;

                //As a safety measure...
                //Refreshes also the whole stage
                WrappingUp();

            }//else : select nothing - the user has not clicked on an object
            break;

        //default stand for CASE tsNoTool, that is, when no Tool is selected
        default :  //Do nothing
            break;
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::ImageStageMouseMove(TObject *Sender,
      TShiftState Shift, int X, int Y)
{
    //Displays mouse pointer location on stage - not on screen
    ToolBox->ToolBoxStatusBar->Panels->Items[3]->Text = " X: "+IntToStr(X)+"  ;
Y: "+IntToStr(Y);

    if (CanDraw)
    {
        DrawTempShape(Origin, MovePt);    //Erases previous shape
        MovePt = Point(X,Y);
        DrawTempShape(Origin, MovePt);    //Draws current shape
    };

    if (CanDrawFreehand)
    {
```

```
        //Draws freehanded on both off- and onscreen by following the pointer
location
        Workbench->Canvas->LineTo(X, Y);        //Draws off-screen on Workbench
        ImageStage->Canvas->LineTo(X,Y);        //Draws on screen to give feedback
to the user

        if (X < FreeHClipRect.Left)
            //Enlarges the left side of the clip.rect. surrounding the freehand
drawing
            FreeHClipRect.Left = X;
        if (Y < FreeHClipRect.Top)
            //Enlarges the left side of the clip.rect. surrounding the freehand
drawing
            FreeHClipRect.Top = Y;
        if (X > FreeHClipRect.Right)
            //Enlarges the left side of the clip.rect. surrounding the freehand
drawing
            FreeHClipRect.Right = X;
        if (Y > FreeHClipRect.Bottom)
            //Enlarges the left side of the clip.rect. surrounding the freehand
drawing
            FreeHClipRect.Bottom = Y;
    };

    if (CanMove)    //Bool for the [Move] function
    {
        if (ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage - Front
View")
        {//[Move in FRONT VIEW]
            //Sets previous coordinates to indicate where was the object
            StageObject->PrevPos_FRONT  = StageObject->NewPos_FRONT;

            //Calculates new coordinates to indicate where to draw the object
            StageObject->NewPos_FRONT = GetObjectNewCoord_FRONT(X, Y, XOffset,
YOffset);

            //Merges NewPos_FRONT and PrevPos_FRONT to get a bigger TRect
including both
            MergeRects_FRONT();

            //Erases everything including the object on its prev. pos. and rebuild
everything including the obj. on its. new pos.
            Rebuild_FRONT(dmMergedClipRect);
        }
        else //[Move in TOP VIEW]
        {
            //Sets previous coordinates to indicate where was the TopRect
            StageObject->PrevPos_TOP  = StageObject->NewPos_TOP;

            //Sets new coordinates to indicate where to draw the TopRect
            StageObject->NewPos_TOP = GetObjectNewCoord_TOP(X, Y, XOffset,
YOffset);

            //Merges NewPos_FRONT and PrevPos_FRONT to get a bigger TRect
including both
            MergeRects_TOP();

            //Erases everything including the object on its prev. pos. and rebuild
everything including the obj. on its. new pos.
            Rebuild_TOP(dmMergedClipRect);
        }
    }
}
//---------------------------------------------------------------------------
```

```
void __fastcall TMainForm::ImageStageMouseUp(TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)
{
    if (CanDraw)
    {
        //Stores the final shape
        StoreShape(Origin, Point(X,Y));

        //Calls the method / member function that rebuilds the StageObjects
overlapping the clip.rect. of the modified object
        Rebuild_FRONT(dmObjectClipRect_FrontView);

        //As a safety measure...
        WrappingUp();

        CanDraw = false;
    };

    if (CanDrawFreehand)
    {
        //Allocates a new memory space for that object on stage and returns a
pointer to it
        StageObject = new TStageObject;
        //Gives an ID number to the object
        StageObject->ID = ++LastObjectID;

        //Positions StorageBmp
        StageObject->NewPos_FRONT = FreeHClipRect;

        //The bitmap's height and width properties hold the original size of the
object
        StageObject->StorageBmp->Width  = FreeHClipRect.Right  -
FreeHClipRect.Left;
        StageObject->StorageBmp->Height = FreeHClipRect.Bottom -
FreeHClipRect.Top;

        //When an object is created, its current dimensions are identical to its
original dimensions
        StageObject->CurWidth  = StageObject->StorageBmp->Width;
        StageObject->CurHeight = StageObject->StorageBmp->Height;

        //Records the center in VA's 3D-space
        StageObject->Center_3D.x = StageObject->NewPos_FRONT.Left + (StageObject-
>StorageBmp->Width  / 2);
        StageObject->Center_3D.y = StageObject->NewPos_FRONT.Top  + (StageObject-
>StorageBmp->Height / 2);
        StageObject->Center_3D.z = 0;

        //Stores the final freehand drawing by copying pixels inside FreeHClipRect
onto StorageBmp at position [0,0]
        StageObject->StorageBmp->Canvas->CopyMode = cmSrcCopy;
        StageObject->StorageBmp->Canvas->CopyRect( Rect(0, 0, StageObject-
>StorageBmp->Width, StageObject->StorageBmp->Height), Workbench->Canvas,
FreeHClipRect);

        //Adds StageObject to the list of objects present on stage
        ListOfObjects->Add(StageObject);

        //Sets [ObjectToSelect] to make it reference to the object that has just
been added to the list
        ObjectToSelect = (ListOfObjects->Count) - 1;       //It's the last item
because the list hasn't been sorted since
```

```
        //Calls the method / member function that rebuilds the StageObjects
overlapping the clip.rect. of the modified object
        Rebuild_FRONT(dmObjectClipRect_FrontView);

        //As a safety measure...
        WrappingUp();

        CanDrawFreehand = false;
    };

    if (CanMove)
    {
        CanMove = false;

        if(ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage - Top
View")
        {//The current view is TOP VIEW

            //Sorts the StageObjects stored in the TList according to their Z
coordinate
            SortStageObjects();

            //Stretches or shrinks the object's ClipRect on the screen's surface
for the FRONT VIEW if we move the object respectively forward or backward along
the z coord
            StageObject->PrevPos_FRONT = StageObject->NewPos_FRONT =
DepthRendering();
        }
        else    //Do nothing special - just comment on the reason for which
nothing is done
        {
            //In FRONT VIEW, the [Sorting]        is done after each step of the
[MoveDeep] loop and not here
            //In FRONT VIEW, the [DepthRendering] is done after each step of the
[MoveDeep] loop and not here
        };

        //As a safety measure...
        WrappingUp();
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::SortStageObjects()
{
/*****************************************************************************
**************************/
/* This function sorts the StageObjects in the TList according to their z
coordinate in VA's 3D space.      */
/*      A decreasing order is used to have at the beginning of the list the
objects which are at            */
/*      the back of the stage, and at the end of the list the objects which are at
the front of             */
/*      the stage.  Simply going through the list and displaying the objects one
by one will allow to       */
/*      keep at the front of the stage the objects with the smallest z
coordinates.  Over and above,        */
/*      when an object is created, it is placed at the front of the stage with its
z coord. equal to        */
/*      zero.  So, placing each object newly created at the end of the list avoid
sorting the list to       */
/*      keep it ordered - [TList::Add(void * Item)] places the item at the end of
the TList.                */
```

```
/* Sorting is fairly easy : when [Moving], only one object at a time is not
ordered - that object is        */
/*      the one that has been selected and is referenced by [int ObjectToSelect].
So first delete that     */
/*      entry from the list.  Then loop through the list till you reach the place
where to place that       */
/*      object.  To do it, use [TList::Insert()] or [TList::Add()] according to
the context - but not       */
/*      [TList::Move()] which sometimes puts the object down one place before or
after the one expected.    */
/* Sorting has to be processed at the end of [Move] in TOP VIEW (i.e. OnMouseUp
event) and after every      */
/*      step of the [Move Deep] loop in FRONT VIEW (i.e. OnMouseMove, boolean
CanMoveDeep).                */
/*
*/
/* The current implementation permits to avoid to be out of bounds.
*/
/*************************************************************************
*************************/

    //[ptrSelectedStageObject] points to the object that has been selected to be
moved
    TStageObject* ptrSelectedStageObject = (TStageObject*) ListOfObjects-
>Items[ObjectToSelect];

    //Removes the StageObject whose z coordinate has been modified
    ListOfObjects->Delete(ObjectToSelect);

    //The index used to loop through the list. The last value of index [i] when
leaving the loop will give us the position in the list where to place our item
    int i;

    //Determines where (in the list) to replace the StageObject whose z coordinate
has been modified
    for (i=0; i < ListOfObjects->Count; i++)
    {
        //[ptrStageObjectAtCurIndex] points to the object referenced by index [i]
in TList::Items[]
        TStageObject* ptrStageObjectAtCurIndex = (TStageObject*) ListOfObjects-
>Items[i];

        if ( ptrSelectedStageObject->Center_3D.z == ptrStageObjectAtCurIndex-
>Center_3D.z )
        {
            if ( ptrSelectedStageObject->ID < ptrStageObjectAtCurIndex->ID )
            {//We have the new position for [SelectedStageObject]
                break;
            }
        }
        else
        {
            if ( ptrSelectedStageObject->Center_3D.z > ptrStageObjectAtCurIndex-
>Center_3D.z )
            {//We have the new position for [SelectedStageObject]
                break;
            }
        }
    }

    if (i == ListOfObjects->Count)
    {//The StageObject whose z coordinate has been modified is placed at the end
of the list
        ListOfObjects->Add(ptrSelectedStageObject);
```

```
        //Sets [ObjectToSelect] to make it reference to the object that has just
been added to the list
        ObjectToSelect = (ListOfObjects->Count) - 1;       //It's the last item
because TList::Add() places its object (the param.) at the end of the list
    }
    else
    {//The StageObject whose z coordinate has been modified is inserted in the
list between two other items
        ListOfObjects->Insert(i, ptrSelectedStageObject);

        //Sets [ObjectToSelect] to make it reference to the object that has just
been added to the list
        ObjectToSelect = i;     //It's [i] because the object has been inserted at
the i-th position
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::WrappingUp()
{
/*****************************************************************************
**************************/
/* This function makes sure that, if the last operation has resulted in a bug, we
can see                    */
/*      it immediately and not after three or four other operations that didn't
resulted                    */
/*      in any bug at all ! For this, the whole World is rebuild.
*/
/* Also, the position of the ClipRects in the other view (the one not displayed at
the moment) is             */
/*      updated here. It indicates as well that the world has been modified and
needs to be saved           */
/*      as well as it activates/desactivates the functionalities that need it.
*/
/*
*/
/* N.B: [ Workbench->Height -1 ] is used because the 522-nd line is at the 521-st
position                   */
/*****************************************************************************
**************************/

    if (ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage - Front
View")
    {//The current view is Front View

         //As a safety measure, we rebuild the whole world (FRONT VIEW) after
every operation modifying the World
        Rebuild_FRONT(dmWhole);

        if (StageObject != NULL)
        {
            //Updates the position of the ClipRects placed on Stage above the
polygon in TOP VIEW
            //Maintains [PrevPos_TOP] and [NewPos_TOP] according to [Center_3D] -
without using [NewPos_FRONT] because this one could be modified for perspective
effect
            int HalfWidth = StageObject->StorageBmp->Width  / 2;
            StageObject->PrevPos_TOP = StageObject->NewPos_TOP = Rect(
StageObject->Center_3D.x - HalfWidth, ((Workbench->Height-1) - StageObject-
>Center_3D.z) - 6, StageObject->Center_3D.x + HalfWidth, ((Workbench->Height-1) -
StageObject->Center_3D.z) + 6 );
```

```
        }//ELSE : do not de-reference StageObject to avoid an 'Access Violation
Error'
    }
    else //->Panels->Items[0]->Text == " Stage - Top View"
    {
        //As a safety measure, we rebuild the whole world (TOP VIEW) after every
operation modifying the World
        Rebuild_TOP(dmWhole);

        //The ClipRects in FRONT VIEW are updated in [DepthRendering()] for
getting a perspective effect
    }

    //The world has been modified - a new object has been added or something has
been moved / modified
    WorldModified = true;

    //Makes available the appropriate functionalities
    UpdateFunctionsAvailable();
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::UpdateFunctionsAvailable()
{
/*****************************************************************************
*******/
/* This function tests which changes needs to be performed in terms of functions
*/
/*      and views available, buttons lowered, etc. according to the status of
*/
/*      the world and the views.
*/
/*****************************************************************************
*******/

//----- CHANGE IN VIEW ?
    if (ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage - Front
View")
    {//The current view is Front View

        //So disactivate what is related to it and activate what's related to Top
View
        StageTopView->Enabled               = true;
        ToolBox->ToolBtnTopView->Enabled    = true;
        StageFrontView->Enabled             = false;
        ToolBox->ToolBtnFrontView->Enabled  = false;

        //Activate drawing functions like Circle or Line that needs to be drawn by
dragging the mouse
        //      - which can only be done in Front View
        EditInsertObject->Enabled                   = true;
        ToolBox->ToolBtnInsertObject->Enabled       = true;

        EditShapeRectangle->Enabled                 = true;
        ToolBox->ToolBtnRectangle->Enabled          = true;

        EditShapeCircle->Enabled                    = true;
        ToolBox->ToolBtnCircle->Enabled             = true;

        EditShapeLine->Enabled                      = true;
        ToolBox->ToolBtnLine->Enabled               = true;

        EditShapeFreeHandPencil->Enabled            = true;
        ToolBox->ToolBtnFreehandPencil->Enabled     = true;
```

```
        EditPenSize->Enabled                          = true;
        ToolBox->ToolBtnPenSize->Enabled              = true;

        //Restores the hint property of [Move] and [Move Deep] since they have a
different behavior from the one they've got in TOP VIEW
        ToolBox->ToolBtnMove->Hint     = "Move an object on the vertical plane";
        ToolBox->ToolBtnMoveDeep->Hint = "Move an object on the horizontal plane";

        //[TransparentObject] is enabled if we are in TOP VIEW with at least 1
object on the stage
    }
    else
    {//The current view is Top View

        //So disactivate what is related to it and activate what's related to
Front View
        StageTopView->Enabled                 = false;
        ToolBox->ToolBtnTopView->Enabled      = false;
        StageFrontView->Enabled               = true;
        ToolBox->ToolBtnFrontView->Enabled    = true;

        //Disactivate drawing functions like Circle or Line that needs to be drawn
by dragging the mouse
        //       - which can only be done in Front View
        EditInsertObject->Enabled                     = false;
        ToolBox->ToolBtnInsertObject->Enabled         = false;

        EditShapeRectangle->Enabled                   = false;
        ToolBox->ToolBtnRectangle->Enabled            = false;

        EditShapeCircle->Enabled                      = false;
        ToolBox->ToolBtnCircle->Enabled               = false;

        EditShapeLine->Enabled                        = false;
        ToolBox->ToolBtnLine->Enabled                 = false;

        EditShapeFreeHandPencil->Enabled              = false;
        ToolBox->ToolBtnFreehandPencil->Enabled       = false;

        EditPenSize->Enabled                          = false;
        ToolBox->ToolBtnPenSize->Enabled              = false;

        //Changes the hint property of [Move] and [Move Deep] since they behave
differently and are the same functions in TOP VIEW
        ToolBox->ToolBtnMove->Hint     = "Move an object backwards and forwards";
        ToolBox->ToolBtnMoveDeep->Hint = "Move an object backwards and forwards";

        //[TransparentObject] is not enabled in TOP VIEW.
    }

//----- AT LEAST ONE OBJECT ON THE STAGE ?
    if (ListOfObjects->Count != 0)
    {// >= 1 objects on the stage
    //FILE menu
        FileSaveWorld->Enabled              = true;
        ToolBox->ToolBtnSaveWorld->Enabled = true;

        FileSaveWorldAs->Enabled             = true;
        ToolBox->ToolBtnSaveWorldAs->Enabled = true;

    //STAGE menu
        StageObjectToWorkRoom->Enabled             = true;
        ToolBox->ToolBtnObjectToWorkRoom->Enabled = true;
```

```
        StageClear->Enabled                  = true;
        ToolBox->ToolBtnStageClear->Enabled = true;

        StageCopyAll->Enabled                  = true;
        ToolBox->ToolBtnStageCopyAll->Enabled = true;

    //WORK menu

    //EDIT menu
        EditCleanPixels->Enabled             = true;
        ToolBox->ToolBtnCleanPixels->Enabled = true;

        EditMove->Enabled              = true;
        ToolBox->ToolBtnMove->Enabled  = true;

        EditMoveDeep->Enabled              = true;
        ToolBox->ToolBtnMoveDeep->Enabled  = true;

        EditResizeObject->Enabled              = true;
        ToolBox->ToolBtnResizeObject->Enabled  = true;

        EditFlipHorizontal->Enabled              = true;
        ToolBox->ToolBtnFlipHorizontal->Enabled  = true;

        EditFlipVertical->Enabled              = true;
        ToolBox->ToolBtnFlipVertical->Enabled  = true;

        EditPattern->Enabled              = true;
        ToolBox->ToolBtnPattern->Enabled  = true;

        EditResizePattern->Enabled              = true;
        ToolBox->ToolBtnResizePattern->Enabled  = true;

        if(ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage - Front
View")
        {//The current view is Front View
            EditTransparentObject->Enabled             = true;
            ToolBox->ToolBtnTransparentObject->Enabled = true;
        }
        else //->Panels->Items[0]->Text == " Stage - Top View"
        {
            EditTransparentObject->Enabled             = false;
            ToolBox->ToolBtnTransparentObject->Enabled = false;
        }

        EditColourColourObject->Enabled        = true;
        ToolBox->ToolBtnColourObject->Enabled  = true;

        EditCutObject->Enabled              = true;
        ToolBox->ToolBtnCutObject->Enabled  = true;

        EditCopyObject->Enabled              = true;
        ToolBox->ToolBtnCopyObject->Enabled  = true;

        EditDeleteObject->Enabled             = true;
        ToolBox->ToolBtnDeleteObject->Enabled = true;

        EditObjectToFloor->Enabled              = true;
        ToolBox->ToolBtnObjectToFloor->Enabled  = true;
    }
    else //There are no objects on the Stage
    {
    //FILE menu
```

```
        FileSaveWorld->Enabled              = false;
        ToolBox->ToolBtnSaveWorld->Enabled = false;

        FileSaveWorldAs->Enabled             = false;
        ToolBox->ToolBtnSaveWorldAs->Enabled = false;

    //STAGE menu
        StageObjectToWorkRoom->Enabled            = false;
        ToolBox->ToolBtnObjectToWorkRoom->Enabled = false;

        StageClear->Enabled                 = false;
        ToolBox->ToolBtnStageClear->Enabled = false;

        StageCopyAll->Enabled                 = false;
        ToolBox->ToolBtnStageCopyAll->Enabled = false;

    //WORK menu

    //EDIT menu
        EditCleanPixels->Enabled             = false;
        ToolBox->ToolBtnCleanPixels->Enabled = false;

        EditMove->Enabled              = false;
        ToolBox->ToolBtnMove->Enabled  = false;

        EditMoveDeep->Enabled              = false;
        ToolBox->ToolBtnMoveDeep->Enabled  = false;

        EditResizeObject->Enabled              = false;
        ToolBox->ToolBtnResizeObject->Enabled  = false;

        EditFlipHorizontal->Enabled              = false;
        ToolBox->ToolBtnFlipHorizontal->Enabled  = false;

        EditFlipVertical->Enabled              = false;
        ToolBox->ToolBtnFlipVertical->Enabled  = false;

        EditPattern->Enabled              = false;
        ToolBox->ToolBtnPattern->Enabled  = false;

        EditResizePattern->Enabled              = false;
        ToolBox->ToolBtnResizePattern->Enabled  = false;

        EditTransparentObject->Enabled             = false;
        ToolBox->ToolBtnTransparentObject->Enabled = false;

        EditColourColourObject->Enabled        = false;
        ToolBox->ToolBtnColourObject->Enabled  = false;

        EditCutObject->Enabled              = false;
        ToolBox->ToolBtnCutObject->Enabled  = false;

        EditCopyObject->Enabled              = false;
        ToolBox->ToolBtnCopyObject->Enabled  = false;

        EditDeleteObject->Enabled             = false;
        ToolBox->ToolBtnDeleteObject->Enabled = false;

        EditObjectToFloor->Enabled              = false;
        ToolBox->ToolBtnObjectToFloor->Enabled  = false;
    }
}
//---------------------------------------------------------------------------
```

```
void __fastcall TMainForm::Rebuild_FRONT(TDisplayMode DisplayMode)
{
/*****************************************************************************
*****************************/
/* Rebuild_FRONT draws first a blank stage, then a wireframe.  Eventually, it
draws all the objects             */
/*      listed as being present on a stage.  When VA is launched, if the user
didn't 2x-click on a VA file,     */
/*      the list contains 0 objects and only a blank stage + a wireframe are drawn
- this is used                */
/*      to prepare a new stage set. Everything is drawn off-screen and finally
displayed on screen              */
/*****************************************************************************
*****************************/
//***** Should have a parameter to tell which stage is concerned

    CreateBlankStage_OffScr();
    S_H_Wireframe_OffScr();
    //***** Paint the floor if necess.
    PaintObjects_OffScr(DisplayMode);

    //Copies Workbench or a part of it on the screen - detects if the object is
being moved
    Display_OnScr(DisplayMode);
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::CreateBlankStage_OffScr()
{
/*****************************************************************/
/* Just wipes the stage with the color used for the Stage       */
/*****************************************************************/

    //Sets pen and brush parameters to draw the blank stage
    Workbench->Canvas->Pen->Style   = psSolid;
    Workbench->Canvas->Pen->Mode    = pmCopy;
    Workbench->Canvas->Pen->Color   = StageColor;
    Workbench->Canvas->Brush->Style = bsSolid;
    Workbench->Canvas->Brush->Color = StageColor;

    //Draws a rectangle symbolizing the blank stage
    Workbench->Canvas->Rectangle(0, 0, ImageStage->Width, ImageStage->Height);
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::S_H_Wireframe_OffScr()
{
/*****************************************************************************
**********************/
/* Shows or hides the wires on the stage according to the WireframeMode, with the
illusion of depth    */
/*****************************************************************************
**********************/

    switch (WireframeMode)
    {
        case (wfShow) :
            //Sets drawing properties
            Workbench->Canvas->Pen->Style   = psSolid;
            Workbench->Canvas->Pen->Mode    = pmCopy;
            Workbench->Canvas->Pen->Color   = clLime;
            Workbench->Canvas->Pen->Width   = 1;
            Workbench->Canvas->Brush->Style = bsClear;
            break;
```

```
        case (wfHide) :
            //Sets drawing properties
            Workbench->Canvas->Pen->Style   = psSolid;
            Workbench->Canvas->Pen->Mode    = pmCopy;
            Workbench->Canvas->Pen->Color   = MainForm->StageColor;
            Workbench->Canvas->Pen->Width   = 1;
            Workbench->Canvas->Brush->Style = bsClear;
            break;
    };

//PART I : Determines the location of the wireframe when projected on the screen's
surface (for perspective)

    //Sides of the projection on the screen's surface of the wireframe's rectangle
placed at the back
    int WF_LeftSide, WF_RightSide, WF_BottomSide, WF_TopSide;

    //Left side of the Wireframe with perspective first, right side then (read
comments there -->)
    WF_LeftSide   = ConvertLocation_3Dto2D( ImageStage->Width /2 , 0
, ImageStage->Height -1);      //In TOP VIEW, the Left-Top  corner is at (0
, 0), that is (0                    , ImageStage->Height -1) in VA's 3D-space
    WF_RightSide  = ConvertLocation_3Dto2D( ImageStage->Width /2 , ImageStage-
>Width -1, ImageStage->Height -1);      //In TOP VIEW, the Right-Top corner is at
(ImageStage->Width -1, 0), that is (ImageStage->Width -1, ImageStage->Height -1)
in VA's 3D-space

    //Bottom side of the Wireframe with perspective first, top side then
    //You've got to look at VA's 3D-space from the right side and erect it so that
the top is on the left of the screen and the bottom on the right.
    WF_TopSide    = ConvertLocation_3Dto2D( ImageStage->Height /2 , 0
, ImageStage->Height -1);
    WF_BottomSide = ConvertLocation_3Dto2D( ImageStage->Height /2 , ImageStage-
>Height -1, ImageStage->Height -1);

//PART II : Draws 1 rectangle and 4 lines
    Workbench->Canvas->Rectangle(WF_LeftSide, WF_TopSide, WF_RightSide,
WF_BottomSide);
    Workbench->Canvas->MoveTo(0, 0);
    Workbench->Canvas->LineTo(WF_LeftSide, WF_TopSide);
    Workbench->Canvas->MoveTo(Workbench->Width, 0);
    Workbench->Canvas->LineTo(WF_RightSide, WF_TopSide);
    Workbench->Canvas->MoveTo(0, Workbench->Height);
    Workbench->Canvas->LineTo(WF_LeftSide, WF_BottomSide);
    Workbench->Canvas->MoveTo(Workbench->Width, Workbench->Height);
    Workbench->Canvas->LineTo(WF_RightSide, WF_BottomSide);
}
//---------------------------------------------------------------------------

void  __fastcall TMainForm::PaintObjects_OffScr(TDisplayMode DisplayMode)
{
/*************************************************************************
***************/
/* This method loops through the list of objects present on the stage and draws
their bitmap    */
/*      (embedded in StageObject under the name StorageBmp) on Workbench (off-
screen bitmap)    */
/*      only if they are inside a certain ClipRect that is determined by
DisplayMode.          */
/*************************************************************************
***************/

    //Points to the StageObject currently considered - we need to know which
ClipRect to consider in the [case dmObjectClipRect_FrontView].
```

```
    TStageObject* CurConsObject;

    for(int i = 0; i < ListOfObjects->Count; i++)
    {
        //Extracts the object at the i-th position in the list of objects present
on stage
        StageObject = (TStageObject*) ListOfObjects->Items[i];

        switch (DisplayMode)
        {
            case dmWhole :
                //Displays all objects present on the stage
                Workbench->Canvas->StretchDraw(StageObject->NewPos_FRONT,
StageObject->StorageBmp);
                break;

            case dmObjectClipRect_FrontView :
                //Displays only objects inside the StageObject's clipping
rectangle currently considered to apply an effect on the object

                //Extracts the StageObject currently considered
                CurConsObject = (TStageObject*) ListOfObjects-
>Items[ObjectToSelect];

                if ( IntersectRects(StageObject->NewPos_FRONT, CurConsObject-
>NewPos_FRONT) )
                {
                    //The object is inside the StageObject's clipping rectangle
currently considered, so paint it onto Workbench
                    Workbench->Canvas->StretchDraw(StageObject->NewPos_FRONT,
StageObject->StorageBmp);

                }//ELSE : do not display the object at the i-th position in the
list because the object is not inside the StageObject's clipping rectangle
currently considered
                break;

            case dmMergedClipRect :
                //Displays only objects inside the MergedRectangle used to move an
object
                if ( IntersectRects(StageObject->NewPos_FRONT, MergedRectangle) )
                {
                    //The object is inside MergedRectangle, so paint it onto
Workbench
                    Workbench->Canvas->StretchDraw(StageObject->NewPos_FRONT,
StageObject->StorageBmp);

                }//ELSE : do not display the object at the i-th position in the
list because the object is not inside MergedRectangle
                break;
        }
    }

    if (ObjectToSelect > -1)
    {
        //Restore the object which has been selected and is used for the moment
        StageObject = (TStageObject*) ListOfObjects->Items[ObjectToSelect];

    }//else : select nothing - the user has not clicked on an object
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::Display_OnScr(TDisplayMode DisplayMode)
{
```

```
/*****************************************************************************
**********************/
/* This function determines which part of the off-screen bitmap has to be
displayed                        */
/*      that is to say :    dmWhole                        --> the whole off-screen
is displayed          */
/*                          dmObjectClipRect_FrontView     --> the object only, in
its Front View         */
/*                          dmObjectClipRect_TopView       --> the object only, in
its Top View           */
/*                          dmMergedClipRect               --> the clip. rect
including the object        */
/*                                                             both on its previous
and new positions     */
/*****************************************************************************
**********************/

    //Sets mode used with CopyRect
    ImageStage->Canvas->CopyMode = cmSrcCopy;

    //Displays Workbench on screen according to the DisplayMode
    TRect AreaToDisplay;
    switch (DisplayMode)
    {
        case dmWhole :
            AreaToDisplay = Rect(0, 0, ImageStage->Width, ImageStage->Height);
            break;
        case dmObjectClipRect_FrontView :
            AreaToDisplay = StageObject->NewPos_FRONT;
            break;
        case dmObjectClipRect_TopView :
            //In TOP VIEW, there's a ClipRect surrounding the object's polygon
when it is angled
            AreaToDisplay = StageObject->NewPos_TOP;
            break;
        case dmMergedClipRect :
            AreaToDisplay = MergedRectangle;
            break;
    };
    ImageStage->Canvas->CopyRect(AreaToDisplay , Workbench->Canvas,
AreaToDisplay);
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::Rebuild_TOP(TDisplayMode DisplayMode)
{
/*****************************************************************************
*****************************/
/* Rebuild_TOP draws first a blank floor of a stage, then it draws a polygon to be
able to select an object     */
/*      listed as being present on a stage.  Everything is drawn off-screen and
finally displayed on screen     */
/*****************************************************************************
*****************************/
//***** Should have a parameter to tell which stage is concerned

    CreateBlankFloor_OffScr();
    DrawTopRects_OffScr(DisplayMode);
    //Copies Workbench or a part of it on the screen - detects if the object is
being moved
    Display_OnScr(DisplayMode);
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::CreateBlankFloor_OffScr()
{
```

```
/*******************************************************/
/* Just wipes the stage's floor with its own color     */
/*******************************************************/

    //Sets pen and brush parameters to draw the floor
    Workbench->Canvas->Pen->Style   = psSolid;
    Workbench->Canvas->Pen->Mode    = pmCopy;
    Workbench->Canvas->Pen->Color   = FloorColor;
    Workbench->Canvas->Brush->Style = bsSolid;
    Workbench->Canvas->Brush->Color = FloorColor;

    //Draws a rectangle symbolizing the blank stage
    Workbench->Canvas->Rectangle(0, 0, ImageStage->Width, ImageStage->Height);
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::DrawTopRects_OffScr(TDisplayMode DisplayMode)
{
/*****************************************************************************
**************/
/* This method loops through the list of objects present on the stage and draws a
polygon thus  */
/*      giving a virtual depth to the object in order to grab and move it.
Everything is       */
/*      drawn on Workbench (off-screen bitmap) only if they are inside a certain
ClipRect       */
/*      that is determined by DisplayMode.  Note that it is especially useful when
the object   */
/*      is being angled.
*/
/*****************************************************************************
**************/

    //Sets pen and brush properties to paint with
    Workbench->Canvas->Pen->Style   = psSolid;
    Workbench->Canvas->Pen->Mode    = pmCopy;
    Workbench->Canvas->Pen->Color   = clAqua;
    Workbench->Canvas->Pen->Width   = 1;
    Workbench->Canvas->Brush->Style = bsSolid;
    Workbench->Canvas->Brush->Color = clGray;

    //Points to the StageObject currently considered - we need to know which
ClipRect to consider in the [case dmObjectClipRect_FrontView].
    TStageObject* CurConsObject;

    for(int i = 0; i < ListOfObjects->Count; i++)
    {
        //Extracts the object at the i-th position in the list of objects present
on stage
        StageObject = (TStageObject*) ListOfObjects->Items[i];

        switch (DisplayMode)
        {
            case dmWhole :
                //Displays all objects present on the stage by drawing a polygon
to be able to see the objects from TOP VIEW
                //***** Later on, use the method [POLYGON]
                Workbench->Canvas->Rectangle(StageObject->NewPos_TOP.Left,
StageObject->NewPos_TOP.Top, StageObject->NewPos_TOP.Right, StageObject-
>NewPos_TOP.Bottom);
                break;

            case dmObjectClipRect_TopView :
                //Displays only object's polygons inside the ClipRect of the
StageObject being angled (remember we are in TOP VIEW)
```

```
                //Extracts the StageObject currently considered (being angled)
                CurConsObject = (TStageObject*) ListOfObjects-
>Items[ObjectToSelect];

                if ( IntersectRects(StageObject->NewPos_TOP, CurConsObject-
>NewPos_TOP) )
                {
                    //The object is inside the StageObject's clipping rectangle
currently angled, so draw its polygon onto Workbench
                    //***** Later on, use the method [POLYGON]
                    Workbench->Canvas->Rectangle(StageObject->NewPos_TOP.Left,
StageObject->NewPos_TOP.Top, StageObject->NewPos_TOP.Right, StageObject-
>NewPos_TOP.Bottom);

                }//ELSE : do not display the object at the i-th position in the
list because the object is not inside the StageObject's clipping rectangle
currently considered
                break;

            case dmMergedClipRect :
                //Displays only objects inside the MergedRectangle used to move an
object's polygon
                if ( IntersectRects(StageObject->NewPos_TOP, MergedRectangle) )
                {
                    //The object is inside MergedRectangle, so draw its polygon
onto Workbench
                    //***** Later on, use the method [POLYGON]
                    Workbench->Canvas->Rectangle(StageObject->NewPos_TOP.Left,
StageObject->NewPos_TOP.Top, StageObject->NewPos_TOP.Right, StageObject-
>NewPos_TOP.Bottom);

                }//ELSE : do not display the object at the i-th position in the
list because the object is not inside MergedRectangle
                break;
        }
    }

    if (ObjectToSelect > -1)
    {
        //Restore the object which has been selected and is used for the moment
        StageObject = (TStageObject*) ListOfObjects->Items[ObjectToSelect];

    }//else : select nothing - the user has not clicked on an object
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::DrawTempShape(POINT MouseLocOnClick, POINT
MouseLocOnRelease)
{
/*****************************************************************************/
/* This method gives the user the ability to draw a shape to the size  wanted   */
/*****************************************************************************/

    //Sets pen and brush parameters to draw with
    ImageStage->Canvas->Pen->Style   = psSolid;
    ImageStage->Canvas->Pen->Mode    = pmNotXor;
    ImageStage->Canvas->Pen->Color   = ColorDialog->Color;      //MainForm's data
member used
    ImageStage->Canvas->Pen->Width   = 1;
    ImageStage->Canvas->Brush->Style = bsSolid;
    ImageStage->Canvas->Brush->Color = ColorDialog->Color;      //MainForm's data
member used
```

```
    //Draws according to the type of the tool selected
    switch (CurTool)
    {
        case tsLine :
            ImageStage->Canvas->Pen->Width = CurrentPenSize;
            ImageStage->Canvas->MoveTo(MouseLocOnClick.x, MouseLocOnClick.y);
            ImageStage->Canvas->LineTo(MouseLocOnRelease.x, MouseLocOnRelease.y);
            break;
        case tsRectangle :
            ImageStage->Canvas->Rectangle(MouseLocOnClick.x, MouseLocOnClick.y,
MouseLocOnRelease.x, MouseLocOnRelease.y);
            break;
        case tsCircle :     //Makes square boundaries to get a circle
            //Local variables to get a circle
            int TempWidth, TempHeight;
            TempWidth  = max(MouseLocOnClick.x, MouseLocOnRelease.x) -
min(MouseLocOnClick.x, MouseLocOnRelease.x);
            TempHeight = max(MouseLocOnClick.y, MouseLocOnRelease.y) -
min(MouseLocOnClick.y, MouseLocOnRelease.y);
            //Does it start on the left or right ?
            if (MouseLocOnRelease.x > MouseLocOnClick.x)
            {//Starts on the left

                //Does it start on the top or bottom ?
                if (MouseLocOnRelease.y > MouseLocOnClick.y)
                {//Starts on the top

                    //Normal case : start rendering from the top left corner
                    ImageStage->Canvas->Ellipse(MouseLocOnClick.x,
MouseLocOnClick.y, MouseLocOnClick.x + min(TempWidth, TempHeight),
MouseLocOnClick.y + min(TempWidth, TempHeight));
                }
                else
                {//starts on the bottom

                    //Start rendering from the bottom left corner
                    ImageStage->Canvas->Ellipse(MouseLocOnClick.x,
MouseLocOnClick.y, MouseLocOnClick.x + min(TempWidth, TempHeight),
MouseLocOnClick.y - min(TempWidth, TempHeight));
                }
            }
            else
            {//Starts on the right

                //Does it start on the top or bottom ?
                if (MouseLocOnRelease.y > MouseLocOnClick.y)
                {//Starts on the top

                    //Start rendering from the top right corner
                    ImageStage->Canvas->Ellipse(MouseLocOnClick.x,
MouseLocOnClick.y, MouseLocOnClick.x - min(TempWidth, TempHeight),
MouseLocOnClick.y + min(TempWidth, TempHeight));
                }
                else
                {//starts on the bottom

                    //Start rendering from the bottom right corner
                    ImageStage->Canvas->Ellipse(MouseLocOnClick.x,
MouseLocOnClick.y, MouseLocOnClick.x - min(TempWidth, TempHeight),
MouseLocOnClick.y - min(TempWidth, TempHeight));
                }
            }
            break;
        case tsEraser :
```

```
            if (MessageDlg("Under construction - Not implemented yet !",
mtWarning,
                TMsgDlgButtons() << mbOK, 0) == mrOk)
                //does nothing
                ;
            break;
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::StoreShape(POINT MouseLocOnClick, POINT
MouseLocOnRelease)
{
/*****************************************************************************
***********/
/* StoreShape creates temporarily a bitmap of the same size as the stage, wipes it
*/
/*      in white and then draws the shape on it at the same position as on the
screen.      */
/*      Finally, the smallest clip. rect. including the shape is copied onto
StorageBmp.    */
/*      To get a circle, it uses Temp- variables to draw an ellipse in square
boundaries    */
/*****************************************************************************
***********/

// PART I : Paints the temporary bitmap in white

    //Creates a temporary bmp
    Graphics::TBitmap* TempBmp = new Graphics::TBitmap;

    //Allocates a new memory space for that object on stage and returns a pointer
to it
    StageObject = new TStageObject;
    //Gives an ID number to the object
    StageObject->ID = ++LastObjectID;

    //Gives StorageBmp the size of ImageStage
    TempBmp->Width  = ImageStage->Width;
    TempBmp->Height = ImageStage->Height;

    //Sets pen and brush properties to paint the bitmap's surface in white first
    TempBmp->Canvas->Pen->Style   = psSolid;
    TempBmp->Canvas->Pen->Mode    = pmCopy;
    TempBmp->Canvas->Pen->Color   = clWhite;
    TempBmp->Canvas->Brush->Style = bsSolid;
    TempBmp->Canvas->Brush->Color = clWhite;
    //Wipes in white
    TempBmp->Canvas->Rectangle(0, 0, TempBmp->Width, TempBmp->Height);

// PART II : Stores (draws) the shape in the temporary bitmap

    //Sets pen and brush color to draw with on TempBmp
    TempBmp->Canvas->Pen->Color   = ColorDialog->Color;
    TempBmp->Canvas->Brush->Color = ColorDialog->Color;
    TempBmp->Canvas->Pen->Width   = 1;

    //Local variables recording the actual size/position of the shape
    int TempShapeWidth, TempShapeHeight, TempShapeLeft, TempShapeTop,
TempShapeRight, TempShapeBottom;

    //At this moment, the size is the one of the rect. ending on last mouse loc.
With a circle, it'll be reduced to the actual size of it.
    TempShapeWidth  = abs(MouseLocOnRelease.x - MouseLocOnClick.x);
```

```
    TempShapeHeight = abs(MouseLocOnRelease.y - MouseLocOnClick.y);

    //At this moment, the position is the one of the rect. ending on last mouse
loc.  With a circle, it'll be set to the actual pos. of it.
    TempShapeLeft   = min(MouseLocOnClick.x, MouseLocOnRelease.x);
    TempShapeTop    = min(MouseLocOnClick.y, MouseLocOnRelease.y);
    TempShapeRight  = max(MouseLocOnClick.x, MouseLocOnRelease.x);
    TempShapeBottom = max(MouseLocOnClick.y, MouseLocOnRelease.y);

    //Draws according to the type of the tool selected
    switch (CurTool)
    {
        case tsLine :
            TempBmp->Canvas->Pen->Width = CurrentPenSize;
            TempBmp->Canvas->MoveTo(MouseLocOnClick.x, MouseLocOnClick.y);
            TempBmp->Canvas->LineTo(MouseLocOnRelease.x, MouseLocOnRelease.y);
            break;
        case tsRectangle :
            TempBmp->Canvas->Rectangle(MouseLocOnClick.x, MouseLocOnClick.y,
MouseLocOnRelease.x, MouseLocOnRelease.y);
            break;
        case tsCircle :
            //Does it start on the left or right ?
            if (MouseLocOnRelease.x > MouseLocOnClick.x)
            {//Starts on the left

                //Does it start on the top or bottom ?
                if (MouseLocOnRelease.y > MouseLocOnClick.y)
                {//Starts on the top

                    //Normal case : starts rendering from the top left corner
                    TempBmp->Canvas->Ellipse(MouseLocOnClick.x, MouseLocOnClick.y,
MouseLocOnClick.x + min(TempShapeWidth, TempShapeHeight), MouseLocOnClick.y +
min(TempShapeWidth, TempShapeHeight));
                    //Records the actual position of the circle, not the one of
the clip.rect. determined by MouseLocOnClick and MouseLocOnRelease
                    TempShapeLeft   = MouseLocOnClick.x;
                    TempShapeTop    = MouseLocOnClick.y;
                    TempShapeRight  = MouseLocOnClick.x + min(TempShapeWidth,
TempShapeHeight);
                    TempShapeBottom = MouseLocOnClick.y + min(TempShapeWidth,
TempShapeHeight);

                }
                else
                {//starts on the bottom

                    //Starts rendering from the bottom left corner
                    TempBmp->Canvas->Ellipse(MouseLocOnClick.x, MouseLocOnClick.y,
MouseLocOnClick.x + min(TempShapeWidth, TempShapeHeight), MouseLocOnClick.y -
min(TempShapeWidth, TempShapeHeight));
                    //Records the actual position of the circle, not the one of
the clip.rect. determined by MouseLocOnClick to MouseLocOnRelease
                    TempShapeLeft   = MouseLocOnClick.x;
                    TempShapeTop    = MouseLocOnClick.y - min(TempShapeWidth,
TempShapeHeight);
                    TempShapeRight  = MouseLocOnClick.x + min(TempShapeWidth,
TempShapeHeight);
                    TempShapeBottom = MouseLocOnClick.y;
                }
            }
            else
            {//Starts on the right
```

```
                //Does it start on the top or bottom ?
                if (MouseLocOnRelease.y > MouseLocOnClick.y)
                {//Starts on the top

                    //Starts rendering from the top right corner and records the
actual size of the circle
                    TempBmp->Canvas->Ellipse(MouseLocOnClick.x, MouseLocOnClick.y,
MouseLocOnClick.x - min(TempShapeWidth, TempShapeHeight), MouseLocOnClick.y +
min(TempShapeWidth, TempShapeHeight));
                    //Records the actual position of the circle, not the one of
the clip.rect. determined by MouseLocOnClick to MouseLocOnRelease
                    TempShapeLeft   = MouseLocOnClick.x - min(TempShapeWidth,
TempShapeHeight);
                    TempShapeTop    = MouseLocOnClick.y;
                    TempShapeRight  = MouseLocOnClick.x;
                    TempShapeBottom = MouseLocOnClick.y + min(TempShapeWidth,
TempShapeHeight);
                }
                else
                {//starts on the bottom

                    //Starts rendering from the bottom right corner and records
the actual size of the circle
                    TempBmp->Canvas->Ellipse(MouseLocOnClick.x, MouseLocOnClick.y,
MouseLocOnClick.x - min(TempShapeWidth, TempShapeHeight), MouseLocOnClick.y -
min(TempShapeWidth, TempShapeHeight));
                    //Records the actual position of the circle, not the one of
the clip.rect. determined by MouseLocOnClick to MouseLocOnRelease
                    TempShapeLeft   = MouseLocOnClick.x - min(TempShapeWidth,
TempShapeHeight);
                    TempShapeTop    = MouseLocOnClick.y - min(TempShapeWidth,
TempShapeHeight);
                    TempShapeRight  = MouseLocOnClick.x;
                    TempShapeBottom = MouseLocOnClick.y;
                }
            }
            //Records the actual size of the circle rather than measuring the
distance to the last mouse loc. given by MouseLocOnRelease
            TempShapeWidth = TempShapeHeight = min(TempShapeWidth,
TempShapeHeight);
            break;
    }

// PART III : Copies the smallest clip. rect. including the shape and records
//            the position of the object on screen in the object's data structure.

    //The bitmap's height and width properties hold the original size of the
object
    StageObject->StorageBmp->Width  = TempShapeWidth;
    StageObject->StorageBmp->Height = TempShapeHeight;

    //When an object is created, its current dimensions are identical to its
original dimensions
    StageObject->CurWidth  = StageObject->StorageBmp->Width;
    StageObject->CurHeight = StageObject->StorageBmp->Height;

    //Recordsd its position on screen
    StageObject->NewPos_FRONT.Left   = TempShapeLeft;
    StageObject->NewPos_FRONT.Top    = TempShapeTop;
    StageObject->NewPos_FRONT.Right  = TempShapeRight;
    StageObject->NewPos_FRONT.Bottom = TempShapeBottom;

    //Records the center in VA's 3D-space
```

```
    StageObject->Center_3D.x = StageObject->NewPos_FRONT.Left + (StageObject-
>StorageBmp->Width  / 2);
    StageObject->Center_3D.y = StageObject->NewPos_FRONT.Top  + (StageObject-
>StorageBmp->Height / 2);
    StageObject->Center_3D.z = 0;

    //Copies pixels at NewPos_FRONT in TempBmp onto StorageBmp at position [0,0]
    StageObject->StorageBmp->Canvas->CopyMode = cmSrcCopy;
    StageObject->StorageBmp->Canvas->CopyRect( Rect(0, 0, StageObject->StorageBmp-
>Width, StageObject->StorageBmp->Height), TempBmp->Canvas, StageObject-
>NewPos_FRONT);
    delete TempBmp;

    //Adds StageObject to the list of objects present on stage
    ListOfObjects->Add(StageObject);

    //Sets [ObjectToSelect] to make it reference to the object that has just been
added to the list
    ObjectToSelect = (ListOfObjects->Count) - 1;       //It's the last item
because the list hasn't been sorted since
}
//---------------------------------------------------------------------------

__fastcall TStageObject::TStageObject()     //Constructor
{
    StorageBmp = new Graphics::TBitmap;
    StorageBmp->TransparentColor = clWhite;     //Sets the color used to apply the
transparency effect
    StorageBmp->Transparent = true;             //Any object's white pixels are
not visible when the object is created
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::MergeRects_FRONT()
{
/*****************************************************************************/
/* This member function (method) takes two rects and records a Merged rect    */
/*      that includes both preceding - used for FRONT VIEW                    */
/* For the moment, we don't pass any rects as parameters, we use data members */
/*      even though this function could be more general - which is not useful */
/*      as far as VA is not concerned.                                        */
/*****************************************************************************/

    if(ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage - Front View")
    {
        MergedRectangle.Left     = min(StageObject->PrevPos_FRONT.Left,
StageObject->NewPos_FRONT.Left);
        MergedRectangle.Top      = min(StageObject->PrevPos_FRONT.Top,
StageObject->NewPos_FRONT.Top);
        MergedRectangle.Right    = max(StageObject->PrevPos_FRONT.Right,
StageObject->NewPos_FRONT.Right);
        MergedRectangle.Bottom   = max(StageObject->PrevPos_FRONT.Bottom,
StageObject->NewPos_FRONT.Bottom);
    }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::MergeRects_TOP()
{
/*****************************************************************************/
/* This member function (method) takes two rects and records a Merged rect    */
/*      that includes both preceding - used for TOP VIEW                      */
/* For the moment, we don't pass any rects as parameters, we use data members */
/*      even though this function could be more general - which is not useful */
```

```
/*      as far as VA is not concerned.                                        */
/*****************************************************************************/

    if (ToolBox->ToolBoxStatusBar->Panels->Items[0]->Text == " Stage - Top View")
    {
        MergedRectangle.Left     = min(StageObject->PrevPos_TOP.Left,
StageObject->NewPos_TOP.Left);
        MergedRectangle.Top      = min(StageObject->PrevPos_TOP.Top,
StageObject->NewPos_TOP.Top);
        MergedRectangle.Right    = max(StageObject->PrevPos_TOP.Right,
StageObject->NewPos_TOP.Right);
        MergedRectangle.Bottom   = max(StageObject->PrevPos_TOP.Bottom,
StageObject->NewPos_TOP.Bottom);
    }
}
//---------------------------------------------------------------------------

bool __fastcall TMainForm::IntersectRects(TRect ComparedRect, TRect
ModelOfComparison)
{
/*****************************************************************************
******************/
/* This member function (method) takes two rects given as parameters and tells if
they intersect    */
/*      one another.  When this method is called from inside a loop, a different
[ComparedRect]     */
/*      is taken and the method checks if it is inside the TRect that serves as
the model of        */
/*      the comparison (sort of a reference, in other words) as it is the same
throughout the       */
/*      calling loop.
*/
/*****************************************************************************
******************/

    if ( ComparedRect.Right  < ModelOfComparison.Left   ||
         ComparedRect.Left   > ModelOfComparison.Right  ||
         ComparedRect.Top    > ModelOfComparison.Bottom ||
         ComparedRect.Bottom < ModelOfComparison.Top      )
    {
        //The two rectangles DO NOT share some pixels - ComparedRect is OUTSIDE
ModelOfComparison
        return false;
    }
    else
    {
        //The two rectangles share some pixels - ComparedRect is INSIDE
ModelOfComparison
        return true;
    };
}
//---------------------------------------------------------------------------

TRect __fastcall TMainForm::GetObjectNewCoord_FRONT(int X, int Y, int XOffset, int
YOffset)
{
/*****************************************************************************
*********************/
/* This function first calculates a possible position of the ClipRect on the
screen, although it      */
/*      may be not valid.  Then, that 2D-location is converted into a VA's 3D-
space location before   */
/*      [CheckBounds()] makes sure the StageObject is within VA's 3D-space bounds.
Finally,          */
```

```
/*      the 3D-location is converted back to a 2D-location.
*/
/* The result returned is the new (valid) location of the StageObject being moved
in FRONT view.      */
/*****************************************************************************
*********************/

//STEP 1 : Calculate a possible location on screen (not necessarily valid) of the
object's projection

    TRect DestClipRect;

    //Records the possible location using the mouse pointer location and the
distance from it to the left and top sides
    DestClipRect.Left   = (X - XOffset);
    DestClipRect.Top    = (Y - YOffset);
    DestClipRect.Right  = (X - XOffset) + StageObject->CurWidth;
    DestClipRect.Bottom = (Y - YOffset) + StageObject->CurHeight;

//STEP 2 : Convert the 2D-location (on screen) into a 3D-location (in VA's 3D-
space)

    int MiddleLine;     //Vertical middle of the view
    TRect Location_3D;  //TRect holding the location of each sides in VA's 3D-
space

    //Converts the left side of the ClipRect first, the right side then
    MiddleLine = ImageStage->Width / 2;
    Location_3D.Left  = ConvertLocation_2Dto3D( MiddleLine, DestClipRect.Left ,
StageObject->Center_3D.z);
    Location_3D.Right = ConvertLocation_2Dto3D( MiddleLine, DestClipRect.Right,
StageObject->Center_3D.z);

    //Bottom side of the ClipRect with perspective first, top side then
    //You've got to look at VA's 3D-space from the right side and make it rotate
so that the top is on the left of the screen and the bottom on the right.
    MiddleLine = ImageStage->Height / 2;
    Location_3D.Top    = ConvertLocation_2Dto3D( MiddleLine, DestClipRect.Top   ,
StageObject->Center_3D.z);
    Location_3D.Bottom = ConvertLocation_2Dto3D( MiddleLine, DestClipRect.Bottom,
StageObject->Center_3D.z);

//STEP 3 : Make location in VA's 3D-space valid and work out [Center_3D]
    CheckBounds(Location_3D);

//STEP 4 : Convert the 3D-location (in VA's 3D-space) into a 2D-location (on
screen) and return result

    //To make the conversion, we just need a valid Center_3D in VA's 3D-space -
that task has been done during STEP 3.
    DestClipRect = DepthRendering();

    return DestClipRect;    //The location on scren is now valid
}
//---------------------------------------------------------------------------

TRect __fastcall TMainForm::GetObjectNewCoord_TOP(int X, int Y, int XOffset, int
YOffset)
{
/*****************************************************************************
**************************/
/* This function first calculates a possible position of the ClipRect on the
screen, although it          */
```

```
/*      may be not valid.  Then, it makes sure that the 'polygon' is within the
bounds in TOP view.          */
/*      The result returned is the new (valid) location of the polygon being moved
in TOP view.              */
/* "Within the bounds" in TOP VIEW means that we can have 6 pixels drawn outside
of the bounds              */
/*    at the top and the bottom of the screen area representing the stage to allow
placing                   */
/*    the [Center_3D.z] exactly at the back of the stage (i.e. the top of the
screen area representing      */
/*    the stage, on 0) or at the front of the stage (i.e. the bottom of the screen
area representing         */
/*    the stage, on [ImageStage->Height-1] ).
*/
/* The X and Z coordinates of [Center_3D] are updated but not the Y because its
irrelevant in TOP VIEW      */
/*    For the X coord., we use the original size of the object because in TOP
VIEW, the polygons            */
/*    representing the object are not subjected to any perspective effect that
could shrink their            */
/*    ClipRect, their width.
*/
/*
*/
/* N.B. : @ "offset towards the top" means 'towards the top of the screen area
representing the stage'.      */
/*          It's an offset towards the 'back of the stage', actually.
*/
/*        @ [ ImageStage->Height-1 ] is used because the 522-nd line is at the
521-st position                */
/*        @ See comment at the top of this file to remember how [Center_3D.z] is
handled.                     */
/*        @ [Y - YOffset] is actually object's [Center_3D.z], that's why we use [Y
- YOffset] to set center */
/*****************************************************************************
**************************/

//STEP 1 : Calculate a possible location in TOP view (not necessarily valid) of
the object

    TRect DestClipRect;

    //Records the possible location using the mouse pointer location and the
distance from it to the left and top sides
    DestClipRect.Left   = (X - XOffset);
    DestClipRect.Top    = (Y - YOffset) - 6;
    DestClipRect.Right  = (X - XOffset) + StageObject->StorageBmp->Width;
    DestClipRect.Bottom = (Y - YOffset) + 6;

//STEP 2 : Make location valid and work out [Center_3D]

  //PART I : left and right sides first considered

    //Makes valid the left and right sides
    if (DestClipRect.Left < 0)
    {
        //Shifts left and right sides to a valid position - offset towards the
right
        DestClipRect.Left  = 0;
        DestClipRect.Right = 0 + StageObject->StorageBmp->Width;
    }
    else
    {
        if (DestClipRect.Right > (ImageStage->Width - 1))
```

```
        {
            //Shifts left and right sides to a valid position - offset towards the
left
            DestClipRect.Left  = (ImageStage->Width - 1) - StageObject-
>StorageBmp->Width;
            DestClipRect.Right =  ImageStage->Width - 1;
        }
        //ELSE : Left and right sides are within bounds and don't need to be
shifted
    }
    //Checks again the left side in the case the object could be larger than the
stage itself.
    //  If so, only the left side is modified - it's like shrinking the ClipRect.
    if (DestClipRect.Left < 0)
    {
        //Shrinks the ClipRect by displacing left side to a valid position -
offset towards the right
        DestClipRect.Left = 0;
    }

    //Works out [Center_3D] : only X is concerned.  Z is modified later on and Y
is left unchanged by [tsMove] in TOP
    //  If the size of the StageObject is larger than the stage, we can't use
[StageObject->StorageBmp->Width /2]
    int HalfWidth = (DestClipRect.Right - DestClipRect.Left) /2;
    StageObject->Center_3D.x = DestClipRect.Left + HalfWidth;

  //PART II : then, [Center_3D.z] is considered - and not top and bottom anymore -
to allow placing it on the top/bottom edge of the stage.
  //    Only Z is concerned.  X is modified here above and Y is left unchanged by
[tsMove] in TOP view

    //Makes valid the [Center_3D.z]
    if ((Y - YOffset) < 0)
    {
        //Shifts [Center_3D.z] to a valid position - offset towards the bottom
        StageObject->Center_3D.z = ImageStage->Height - 1;  //[Center_3D.z] is at
the back of the Stage
    }
    else
    {
        if ((Y - YOffset) > (ImageStage->Height - 1))
        {
            //Shifts top and bottom sides to a valid position - offset towards the
top
            StageObject->Center_3D.z = 0;                    //[Center_3D.z] is at
the front of the Stage
        }
        else //[Center_3D.z] is within bounds and don't need to be shifted
        {
            StageObject->Center_3D.z = (ImageStage->Height - 1) - (Y - YOffset);
        }
    }
    //Don't need to check again something because in TOP, the polygon's height
can't be taller than the stage itself.

    //Works out the top and bottom sides of [DestClipRect]
    DestClipRect.Top    = (ImageStage->Height - 1) - StageObject->Center_3D.z - 6;
    DestClipRect.Bottom = (ImageStage->Height - 1) - StageObject->Center_3D.z + 6;

    return DestClipRect;    //The location on scren is now valid
}
//---------------------------------------------------------------------------
```

```
TRect __fastcall TMainForm::DepthRendering()
{
/*****************************************************************************
**************************/
/* This function calls 4 times [ConvertLocation_3Dto2D]. The first time, we
consider a top view and        */
/*      we pass the coord. for the left and right sides of the object like if it
was seen from top.          */
/*      Then we pass the coord. of the bottom and top sides like if we were in a
side view - which           */
/*      doesn't actually exist. We also pass a [MiddleLine] that runs from the
back of the stage to         */
/*      the front in both views - although they are not placed on the same spot.
*/
/* Called 4 times, that triangle function gives us the TRect of the object's
projection on the screen's      */
/*      surface needed to get a perspective effect.  We just have to record the
current size of the          */
/*      object because it can be different from its original size if there's a
perspective effect.          */
/*
*/
/* N.B: The 3rd parameter of [ConvertLocation_3Dto2D()] is [StageObject-
>Center_3D.z] and not                */
/*      [(ImageStage->Height -1) - StageObject->Center_3D.z] because we consider a
distance (D1)             */
/*      that runs from the user's eye to the screen (600 units) and from the
screen to the object           */
/*      (given by [->Center_3D.z]) - so we don't need to take [ImageStage->Height
- 1] from which           */
/*      we substract [Center_3D.z] to get the displacement from the top of the
screen.                      */
/*****************************************************************************
**************************/

    int MiddleLine, HalfWidth;

    //It is the ClipRect of an object projected on the screen's surface (for
perspective effect) when this object is moved to the back of the screen
    TRect ClipRectProjectedOnScr;

//Works out the location on the screen's surface (for perspective effect) of the
ClipRect of an object which is moved to the back of the screen
    //Left side of the ClipRect with perspective first, right side then
    MiddleLine = ImageStage->Width / 2;
    HalfWidth  = StageObject->StorageBmp->Width  / 2;
    ClipRectProjectedOnScr.Left   = ConvertLocation_3Dto2D( MiddleLine,
StageObject->Center_3D.x - HalfWidth, StageObject->Center_3D.z);
    ClipRectProjectedOnScr.Right  = ConvertLocation_3Dto2D( MiddleLine,
StageObject->Center_3D.x + HalfWidth, StageObject->Center_3D.z);

    //Bottom side of the ClipRect with perspective first, top side then
    //You've got to look at VA's 3D-space from the right side and make it rotate
so that the top is on the left of the screen and the bottom on the right.
    MiddleLine = ImageStage->Height / 2;
    HalfWidth  = StageObject->StorageBmp->Height / 2;
    ClipRectProjectedOnScr.Top    = ConvertLocation_3Dto2D( MiddleLine,
StageObject->Center_3D.y - HalfWidth, StageObject->Center_3D.z);
    ClipRectProjectedOnScr.Bottom = ConvertLocation_3Dto2D( MiddleLine,
StageObject->Center_3D.y + HalfWidth, StageObject->Center_3D.z);

    //The object might currently have a size different from its original one
    StageObject->CurWidth  = (ClipRectProjectedOnScr.Right  -
ClipRectProjectedOnScr.Left);
```

```
    StageObject->CurHeight = (ClipRectProjectedOnScr.Bottom -
ClipRectProjectedOnScr.Top);

    //The results given here above are re-used to shrink or stretch the ClipRect -
NOT the bitmap itself which keeps its original size.
    return ClipRectProjectedOnScr;
}
//---------------------------------------------------------------------------

int __fastcall TMainForm::ConvertLocation_3Dto2D(int MiddleLine, int HorizCoord,
int VertCoord)
{
/*****************************************************************************
**************************/
/* When an object is moved backward or forward in VA's 3D space, because the
screen is in 2D we need to     */
/*      shrink or stretch it to give the illusion of depth.  Considering an object
not at the front         */
/*      of the stage, we have to determine its projection on the screen's surface
with an effect of         */
/*      perspective.  For that, we take one side of the object at a time and give
its location              */
/*      horizontally and vertically(^).  To work out the location on the screen,
we use two triangles       */
/*      with a rectangular corner.  The base of the biggest starts on user's
location and ends on the       */
/*      object - named [D1].  Its height is the distance between a middle line
(running from back-stage     */
/*      to front-stage) and the considered side of the object (either left or
right) - named [H1].          */
/*      The base of the smallest starts from the user again and ends on the screen
- named [D2].             */
/*      Its height runs from the middle again to its vertex on the screen's
surface after projection        */
/*      on it - named [H2]; it's the 'unknown'.
*/
/* Knowing that ( H1/D1 ) = ( H2/D2 ), we can calculate H2 and return the distance
from the left            */
/*      of the stage - that is the value for the middle line plus H2.
*/
/*
*/
/* N.B: (^) [VertCoord] starts at the bottom.  D1 runs from the user's eye to the
screen (600 units)        */
/*      and from the screen to the object (given by [VertCoord]) - so we don't
need to take                */
/*      [ImageStage->Height - 1] from which we substract [Center_3D.z] to get the
displacement              */
/*      from the top of the screen.
*/
/*****************************************************************************
**************************/

    int D1 = UserToScreenDist + VertCoord;
    int D2 = UserToScreenDist;
    //If [HorizCoord] is smaller, we know that the point in VA's TOP VIEW is on
the left of the
    //      [MiddleLine] and so the result is negative - will be useful later on.
    float H1 = HorizCoord - MiddleLine;
    //Calculates the 'unknown', the horizontal location on the screen to give the
impression of depth
    float H2 = (H1 / D1) * D2;
```

```
    //If [H2] is negative, the location on the screen's surface is on the left of
the middle line.
    //      If not, it's on the right.  So we just need to add [H2] to the value
representing the middle line
    //      and we get the distance from the left of the stage - just what we need
to display.
    return (MiddleLine + H2);
}
//---------------------------------------------------------------------------

int __fastcall TMainForm::ConvertLocation_2Dto3D(int MiddleLine, int HorizCoord,
int VertCoord)
{
/*****************************************************************************
**************************/
/* It is the inverse of [ConvertLocation_3Dto2D].  We have the ClipRect for the
projection of the           */
/*      StageObject on screen and we need to know where it is placed in VA's 3D-
space.                      */
/* With this aim in view, we take one side of the ClipRect (on scr.) at a time and
give its location        */
/*      horizontally and vertically(^).  To work out the location in 3D-space, we
use two triangles          */
/*      with a rectangular corner.  The base of the biggest starts on user's
location and ends on the        */
/*      object - named [D2].  Its height is the distance between a middle line
(running from back-stage     */
/*      to front-stage) and the considered side of the object (either left or
right) - named [H2].          */
/*      The base of the smallest starts from the user again and ends on the screen
- named [D1].             */
/*      Its height runs from the middle again to its vertex on the screen's
surface after projection         */
/*      on it - named [H1].  [H2] is the 'unknown'.
*/
/* Knowing that ( H1/D1 ) = ( H2/D2 ), we can calculate H2 and return the distance
from the left             */
/*      of the stage - that is the value for the middle line plus H2.
*/
/*
*/
/* N.B: (^) [VertCoord] starts at the bottom.  D1 runs from the user's eye to the
screen (600 units)         */
/*      and from the screen to the object (given by [VertCoord]) - so we don't
need to take                 */
/*      [ImageStage->Height - 1] from which we substract [Center_3D.z] to get the
displacement               */
/*      from the top of the screen.
*/
/*****************************************************************************
**************************/

    int D1 = UserToScreenDist;
    int D2 = UserToScreenDist + VertCoord;
    //If [HorizCoord] is smaller, we know that the point in VA's TOP VIEW is on
the left of the
    //      [MiddleLine] and so the result is negative - will be useful later on.
    float H1 = HorizCoord - MiddleLine;
    //Calculates the 'unknown', the horizontal location on the screen to give the
impression of depth
    float H2 = (H1 / D1) * D2;

    //If [H2] is negative, the location on the screen's surface is on the left of
the middle line.
```

```
    //      If not, it's on the right.  So we just need to add [H2] to the value
representing the middle line
    //      and we get the distance from the left of the stage - just what we need
to display.
    return (MiddleLine + H2);
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::CheckBounds(TRect ValidLocation_3D)
{
/*****************************************************************************
**************************/
/* This function makes sure the StageObject is within VA's 3D-space bounds.  With
this aim in view,         */
/*      it first considers the left and right sides in a top view and offset them
if they are not           */
/*      within the bounds.  It also takes into consideration the fact that an
object can be bigger          */
/*      than the stage itself.  Then, the same work is done for the top and bottom
sides but from           */
/*      a side view this time.
*/
/* For each view taken into consideration, only one coordinate of [Center_3D] is
calculated at              */
/*      a time.  Z is not modified by a normal move in X and Y directions.
*/
/*
*/
/* N.B. : @ "offset towards the top" means 'towards the top of the screen area
representing the stage'.     */
/*          In this case, it's an offset towards the 'actual top of the stage'.
*/
/*        @ [ImageStage->Height -1] is used because the 522-nd  line is at the
521-st  position             */
/*        @ [ImageStage->Width  -1] is used because the 1024-nd line is at the
1023-st position             */
/*****************************************************************************
**************************/

    int HalfWidth;

//PART I : left and right sides first considered

    //Considers a view from the 'TOP' to make valid the left and right sides
    if (ValidLocation_3D.Left < 0)
    {
        //Shifts left and right sides to a valid position - offset towards the
right
        ValidLocation_3D.Left  = 0;
        ValidLocation_3D.Right = 0 + StageObject->StorageBmp->Width;
    }
    else
    {
        if (ValidLocation_3D.Right > (ImageStage->Width - 1))
        {
            //Shifts left and right sides to a valid position - offset towards the
left
            ValidLocation_3D.Left  = (ImageStage->Width - 1) - StageObject-
>StorageBmp->Width;
            ValidLocation_3D.Right =  ImageStage->Width - 1;
        }
        //ELSE : Left and right sides are within bounds and don't need to be
shifted
    }
```

```
    //Checks again the left side in the case the object could be larger than the
stage itself.
    //  If so, only the left side is modified - it's like shrinking the ClipRect.
    if (ValidLocation_3D.Left < 0)
        //Shrinks the ClipRect by displacing left side to a valid position -
offset towards the right
        ValidLocation_3D.Left = 0;

    //Works out [Center_3D] : only X is concerned.  Y is modified later on and Z
is left unchanged by [tsMove]
    //  If the size of the StageObject is larger than the stage, we can't use
[StageObject->StorageBmp->Width /2]
    HalfWidth = (ValidLocation_3D.Right - ValidLocation_3D.Left) /2;
    StageObject->Center_3D.x = ValidLocation_3D.Left + HalfWidth;

//PART II : then, top and bottom considered

    //Considers a view from the 'SIDE' to make valid the top and bottom sides
    if (ValidLocation_3D.Top < 0)
    {
        //Shifts top and bottom sides to a valid position - offset towards the
bottom
        ValidLocation_3D.Top    = 0;
        ValidLocation_3D.Bottom = 0 + StageObject->StorageBmp->Height;
    }
    else
    {
        if (ValidLocation_3D.Bottom > (ImageStage->Height - 1))
        {
            //Shifts top and bottom sides to a valid position - offset towards the
top
            ValidLocation_3D.Top    = (ImageStage->Height - 1) - StageObject-
>StorageBmp->Height;
            ValidLocation_3D.Bottom =  ImageStage->Height - 1;
        }
        //ELSE : Top and bottom sides are within bounds and don't need to be
shifted
    }
    //Checks again the top side in the case the object could be taller than the
stage itself.
    //  If so, only the top side is modified - it's like shrinking the ClipRect.
    if (ValidLocation_3D.Top < 0)
        //Shrinks the ClipRect by displacing top side to a valid position - offset
towards the bottom
        ValidLocation_3D.Top  = 0;

    //Works out [Center_3D] : only Y is concerned.  X is modified here above and Z
is left unchanged by [tsMove]
    //  If the size of the StageObject is taller than the stage, we can't use
[StageObject->StorageBmp->Height /2]
    HalfWidth = (ValidLocation_3D.Bottom - ValidLocation_3D.Top) /2;
    StageObject->Center_3D.y = ValidLocation_3D.Top + HalfWidth;
}
//---------------------------------------------------------------------------
```